

# CUM SA PROGRAMEZI IN NCURSES

De [Pradeep Padala](#)

V.1.3.2 (11/02/2001).

---

*Acest document se doreste a fi un ghid «Tot in Unul » pentru programarea in ncurses si bibliotecile sale inrudite. Progresam de la un program simplu “Hello World” la manipularea unor forme mai complexe. Nu presupune o experienta anterioara in ncurses. Ultima versiune a documentului poate fi gasita intotdeauna pe site-ul meu web. Trimiteti comentarii la aceasta adresa.*

---

## Continut

- [Introducere](#)
  - [Ce este NCURSES](#)
  - [Ce putem face cu NCURSES](#)
  - [De unde se poate lua](#)
  - [Intentia/Scopul acestui document](#)
  - [Despre programe](#)
  - [Credite](#)
  - [Lista de dorinte](#)
  - [Schimbarea inregistrarii](#)
  - [Drepturi de copiere](#)
- [Programul “Hello World”](#)
  - [Compilarea cu librariile Ncurses](#)
  - [Codul](#)
  - [Disecarea](#)
    - [Despre initscr\(\)](#)
    - [Despre misteriosul refresh\(\)](#)
    - [Despre endwin\(\)](#)
- [Detaliile sangeroase](#)
  - [Initializare](#)
  - [Despre functii de initializare ca raw\(\) etc...](#)
    - [raw\(\) si cbreak\(\)](#)
    - [echo\(\) si noecho\(\)](#)
    - [keypad\(\)](#)
    - [halfdelay\(\)](#)
    - [Variate functii de initializare](#)
    - [Un exemplu](#)
  - [Cateva cuvinte despre Windows](#)
  - [Despre functii de iesire ca printf\(\)](#)
    - [clasa de functii printf\(\)](#)
      - [printf\(\) si mvprintf\(\)](#)

- [wprintw\(\) si mvprintw\(\)](#)
    - [vwprintw\(\)](#)
  - [clasa de functii addstr\(\)](#)
  - [precautie](#)
  - [Cateva exemple](#)
- [Depsre functii de intrare ca scanw\(\) etc..](#)
  - [clasa de functii scanw\(\)](#)
    - [scanw\(\) si mvscanw\(\)](#)
    - [wscanw\(\) si mvwscanw\(\)](#)
    - [vwscanw\(\)](#)
  - [clasa de functii getstr\(\)](#)
  - [Cateva exemple](#)
- [Atribute](#)
  - [Exemplu](#)
  - [Detalii](#)
  - [attron\(\) vs attrset\(\)](#)
  - [attr\\_get\(\)](#)
  - [attr functii](#)
  - [wattr functii](#)
  - [chgat\(\) functii](#)
  - [chgat\(\) exemplu](#)
- [Totul despre functiile ferestrelor](#)
  - [Baza](#)
  - [Sa fie o fereastr!!!](#)
  - [Exemplul Border](#)
  - [Explicatie](#)
  - [Celelalte lucruri din exemplu](#)
  - [Alte functii Border](#)
  - [mvhline\(\) si mvvline\(\)](#)
- [Totul despre culori](#)
  - [Un exemplu simplu](#)
  - [Schimbarea definirii culorii](#)
  - [Continutul culorii](#)
- [Folosirea tastelor.Cum sa citesti functiile taste, tastele sageti etc..](#)
  - [Baza](#)
  - [Codul](#)
- [Interactionarea cu mouse-ul](#)
  - [Baza](#)
  - [Primirea evenimentelor](#)
  - [Un exemplu de interactiune cu mouse-ul](#)
  - [Functii diverse](#)
- [Lucrul cu ecranul](#)
  - [getyx\(\)](#)
  - [Salvarea ecranului](#)
  - [Salvarea ferestrei](#)
- [Functii diverse](#)

- [curs\\_set\(\)](#)
  - [Parasirea temporara a modului Curses](#)
- [Alte biblioteci](#)
  - [Libraria Panel](#)
    - [Baza](#)
    - [Compilarea cu librariile Panel](#)
    - [Un exemplu simplu](#)
    - [Un exemplu despre navigarea unei ferestre](#)
    - [Folosirea pointerilor utilizator](#)
    - [Mutarea si redimensionarea panelurilor](#)
    - [Exemplu despre mutare si redimensioanare](#)
    - [Ascunderea si vizualizarea panelurilor](#)
    - [Exemplu despre ascunderea si vizualizarea panelurilor](#)
    - [Functiile `panel\_above\(\)` si `panel\_below\(\)`](#)
  - [Libraria meniurilor](#)
    - [Baza](#)
    - [Compilarea cu librariile meniurilor](#)
    - [Un exemplu simplu](#)
    - [Meniul principal: Baza meniului sistemului](#)
    - [Meniul ferestrelor](#)
    - [Exemplu depre meniul ferestrelor](#)
    - [Derularea meniurilor](#)
    - [Exemplu despre derularea meniurilor](#)
    - [Meniuri multi coloane](#)
    - [Exemplu despre meniuri multi coloana](#)
    - [Meniuri multi valori](#)
    - [Exemplu despre Meniuri multi valori](#)
    - [Optiunile meniurilor](#)
    - [exemplu despre optiunile meniurilor](#)
    - [Utilitatea pointerilor utilizator](#)
    - [Exemplu despre pointerii utilizator](#)
  - [Libraria formularelor](#)
    - [Baza](#)
    - [Compilarea cu librariile formularelor](#)
    - [Un exemplu simplu](#)
    - [Jocul cu campurile](#)
      - [Uimitoarea dimensiune si locatie a campurilor](#)
      - [Deplasarea campului](#)
      - [Alinierea campului](#)
      - [Exmplu despre afisarea atributelor](#)
      - [Biti optionali pentru campuri](#)
      - [Exemplu despre optiunile de camp](#)
      - [Starea campului](#)
      - [Campul pointer utilizator](#)
      - [Campuri cu dimensiune variabila](#)

- [Validarea campurilor](#)
    - [TYPE\\_ALPHA](#)
    - [TYPE\\_ALNUM](#)
    - [TYPE\\_ENUM](#)
    - [TYPE\\_INTEGER](#)
    - [TYPE\\_NUMERIC](#)
    - [TYPE\\_REGEX](#)
  - [Formularul principal:Baza formularelor sistemului](#)
    - [Cereri de navigare paginata](#)
    - [Cereri de navigare intre campuri](#)
    - [Cereri de navigare prin campuri](#)
    - [Cereri de derulare](#)
    - [Cereri de editare](#)
    - [Cereri de ordonare](#)
    - [Comenzi ale aplicatiei](#)
- [Bibliotecile Tools si Widget](#)
  - [CDK \(Curses Development Kit\)](#)
    - [Widget List](#)
    - [Cateva caracteristici interesante](#)
    - [Concluzie](#)
  - [Dialogul](#)
  - [Perl Curses Widgets CURSES::FORM si CURSES:WIDGETS](#)
- [Distractiv](#)
  - [Jocul vietii](#)
  - [Patratul magic](#)
  - [Turnurile din Hanoi](#)
  - [Puzzelul reginelor](#)
  - [Imprastierea](#)
  - [Tutorial de tastare](#)
- [Referinte](#)
  - [Intrebari depre curses](#)
  - [Pagina curses](#)
  - [Programarea in curses](#)
  - [Ghid de programarea in Extended Terminal Interface \(ETI\)](#)

## Introducere

In timpurile mai vechi ale terminalelor telex, terminalele erau departe de caculoare si erau contactate la ele prin cabluri serial. Terminalele puteau fi configurate trimitand o serie de octeti la fiecare din ele. Toate capacitatile( cum ar fi mutarea cursorului la o noua locatie, stergerea unei parti din ecran, parcurgerea ecranului, schimbarea modului, schimbarea aspectului, culorilor, luminozitatii, licaririi, sublinierii, video revers etc.) terminalelor puteau fi accesate prin aceste serii de octeti care sunt numite de obicei secvente escape pentru ca ele incep cu un caractere escape (0x1B). Chiar si azi, cu o simulare potrivita, putem trimite secvente escape simulatorului si obtine acelasi efect la fereastra terminala.

Sa presupunem ca dorim sa afisam o linie colorata. Incercati sa tastati la consola:

```
echo "^[[0;31;40mIn Color"
```

Primul caracter intr-un caracter escape, care seamana cu doua caractere ^ si [. Pentru a putea printa ce aveti nevoie, apasat CTRL+V si apoi ESC key. Toate celelalte caractere sunt normale la printare. Ar trebui sa vezi cuvantul "In Color" in rosu. El ramane asa si pentru a-l converti inapoi la modul original tastati :

```
echo "^[[0;37;40m"
```

Acum, ce inseamna acele caractere magice? Dificil de inteles? Ele pot diferi pentru terminale diferite. Asa ca proiectantii Unix-ului au inventat un mecanism numit termcap. Este un fisier care listeaza toate capacitatile unui terminal particular, impreuna cu secventele escape necesare pentru a obtine un efect particular. In anii urmatori, acesta a fost inlocuit de terminfo. Fara a intra prea mult in detalii, conceptul mecanismului este sa permita programelor aplicatiilor sa intrebe baza de date terminfo si sa obtina caracterle de control care sa fie trimise terminalului sau simulatorului de terminal.

## Ce este NCURSES?

Va intrebati probabil care este rostul acestor discutii tehnice. In scenariul de mai sus, fiecare program aplicatie ar trebui sa interogheze terminfo si sa faca ceea ce trebuie ( sa trimita caractere de control etc.). Curand a devenit complicat sa lucrezi cu acesta complexitate si asa a luat nastere 'CURSES'. Curses este un joc de cuvine plecand de la « optimizarea cursorului ».Libraria Curses formeaza o coperta a lucrului cu coduri neprelucrate ale terminalelor, si furnizeaza un API (Application Programming Interface) foarte flexibil si eficient. Acesta furnizeaza functii pentru mutarea cursorului, crearea de ferestre, producerea de culori, jucarea cu mouse-ul etc. Programele aplicatii nu se preocupa de capacitatile fundamentale ale terminalelor.

Deci ce este NCURSES? NCURSES este o clona a originalului curses System V Release 4.0 (SVr4) . Este o librerie liber distributiva, complet compatibila cu versiunile mai vechi de curses. Pe scurt, acesta este o librerie de functii care administreaza aspectul unei aplicatii pe celulele-caracter ale terminalelor. In continuarea documentului, termenii curses si ncurses se pot inlocui intre ei. Pachetul ncurses a fost creat de Pavel Curtis. Originalul sustinator al acestui pachet este Zeyd Ben-Halim <zmbenhal@netcom.com>. [Eric S. Raymond](mailto:esr@snark.thyrsus.com) <esr@snark.thyrsus.com > a scis multe dintre noile caracteristici in versiunile dupa 1.8.1. [Jürgen Pfeifer](#) a scris tot codul pentru meniuri si formulare, precum si legatura Ada95. In continuare munca a fost facuta de Thomas Dickey si [Jürgen Pfeifer](#). [Florian La Roche](#) reprezinta mentinatorul pentru Free Software Foundation, care detine dreptul de copiere in ncurses. Contactati actualii sustinatori la [bug-ncurses@gnu.org](mailto:bug-ncurses@gnu.org).

## Ce putem face cu NCURSES

Ncurses nu numai ca creaza o coperta peste capacitatile terminalelor, dar ofera si un schelet robust pentru crearea unei interfete placute cu utilizatorul UI in mod text. Furnizeaza functii pentru crearea de ferestre etc. Librariile sale inrudite panel, meniu si formular furnizeaza o extensie a bibliotecii de baza curses. Aceste librarii vin impreuna cu curses. Se pot crea aplicatii care sa contina multiple ferestre, meniuri, paneluri si formulare. Ferestrele pot fi administrate independent, pot furniza "scrollability" si pot fi chiar si ascunse.

Meniurile furnizeaza utilizatorului o optiune facila a comenzii de selectare. Formularele permit crearea a unor date de intrare si de afisare a ferestrelor usor de folosit. Panelurile extind capacitatile pentru ncurses de a gestiona ferestre suprapuse.

Acestea sunt doar cateva dintre lucrurile elementare pe care le putem face cu ncurses. Cu cat avansam, vom vedea toate capacitatile acestor librarii.

## De unde il putem lua

Bine, acum ca stiti ce putem face cu ncurses, trebuie sa incepeti. Ncurses este de obicei in instalare. In cazul in care nu aveti biblioteca sau vreti sa o compilati singuri, citit mai departe.

### *Compilarea pachetului*

Ncurses poate fi obtinut la adresa <ftp://ftp.gnu.org/pub/gnu/ncurses/ncurses.tar.gz> sau la orice site de ftp mentioante in <http://www.gnu.org/order/ftp.html>. Ultima eliberare stabila este 5.2 20001021.

Cititi fisierele pentru detalii despre cum sa instalezi. De obicei implica urmatoarele operatii.

```
gunzip ncurses.tar.gz          # dezarchiveaza fisierul
tar -xvf ncurses.tar          # dezarchiveaza arhiva
./configure                    # configureaza constructia conform cu mediul
make                           # construiește-l
su -root                       # devina root
make install                   # instaleaza-l
```

### *Folosirea RPM-ului*

Ncurses RPM poate fi gasit si descarcat de la <http://rpmfind.net>. RPM-ul poate fi instalat cu urmatoarele comenzi dupa ce ati devenit root.

```
rpm -i <downloaded rpm>
```

## Intentia/Scopul documentului

Acest document se doreste a fi un ghid «Tot in Unul » pentru programarea in ncurses si librariile sale inrudite. Progresam de la un program simplu "Hello World" la manipularea unor forme mai complexe. Nu presupune o experienta anterioara in ncurses.

## Despre programe

Toate programele din document sunt disponibile in format zipp pe site-ul meu. Dezarhiveaza-le. Structura directoarelor arata astfel.

```
ncurses
|
|----> JustForFun    -- programe de destindere
|----> basics        -- programe elementare
|----> demo          -- fisierele de iesire merg in acest director dupa
                    comanda make
                    |----> exe      --fisiere exe pentru toate programele exemplu
                    |----> obj      --fisiere object pentru toate programele
                    exemplu
|----> forms         -- programe legate de librariile formular
|----> menus         -- programe legate de librariile meniu
|----> panels        -- programe legate de librariile panel
|----> Makefile      -- Makefile de nivel inalt
|----> README        -- fisierul README de nivel inalt.Contine
instructiuni
|----> COPYING        -- observatii despre dreptul de copiere
|----> NCURSES_HOWTO.html - acest fisier
```

Directoarele individuale contin urmatoarele fisiere.

Descrierea fisierelor din fiecare director

-----

JustForFun

```
|
|----> hanoi.c       --Rezolvarea Turnurile din Hanoi
|----> life.c        -- Demo Jocul Vietii
|----> magic.c       -- Constructia pentru Odd Order Magic Square
|----> queens.c     -- Rezolvarea faimoasei probleme N-regine
|----> shuffle.c    -- Un joc distractiv daca ai timp liber
|----> tt.c         -- Un tutorial trivial pentru tastare
```

basics

```
|
|----> hello_world.c -- Un program simplu "Hello World"
|----> init_func_example.c -- Exemplu despre functii de initializare
|----> key_code.c    -- Arata codul tastei apasate
|----> mouse_menu.c -- Un meniu accesibil cu mouse-ul
|----> other_border.c -- Arata utilitatea celorlalte functii border
```

deosebite

```
|
|----> printw_example.c -- box()
|----> scanw_example.c  -- Un foarte simplu exemplu printf()
|----> scanw_example.c  -- Un foarte simplu exemplu getstr()
```

```

    |----> simple_attr.c      -- un program care poate afisa un fisier c cu
comentarii                    |
    |                        -- in attribute
    |----> simple_color.c    -- Un exemplu simplu folosind culori
    |----> simple_key.c      --Un meniu accesibil cu tastatura prin
tastele sageti Sus
    |----> temp_leave.c      -- Demonstreaza parasirea temporara a modului
curses
    |----> win_border.c      -- Arata crearea ferestrelor si a
chenarelor
    |----> with_chgat.c      -- Exemplu de folosire a functiei
chgat()
forms
    |
    |----> form_attrib.c      -- Folosirea atributelor camp
    |----> form_options.c    -- Folosirea optiunilor camp
    |----> form_simple.c     -- Un exemplu simplu cu formulare
    |----> form_win.c        -- Exemplu de ferestre asociate cu formulare

menus
    |
    |----> menu_attrib.c     -- Folosirea atributelor de meniu
    |----> menu_item_data.c  -- Folosirea functiilor item_name() etc..
    |----> menu_multi_column.c -- Creaza meniuri multi-coloana
    |----> menu_scroll.c     -- Arata capacitatea de scrolling a
meniurilor
    |----> menu_simple.c     -- Un simplu meniu accesat de tastele sageti
    |----> menu_toggle.c     -- Creaza meniuri multi-valoare si explica
    |                        -- REQ_TOGGLE_ITEM
    |----> menu_userptr.c    -- Folosirea pointerilor utilizator
    |----> menu_win.c        -- Exemplu de fereastră asociată cu meniu

panels
    |
    |----> panel_browse.c    -- Parcurgerea panelului prin tab. Folosirea
pointerilor utilizator
    |----> panel_hide.c      -- Ascunderea si afisarea panelurilor
    |----> panel_resize.c    -- Mutarea si redimensionarea panelurilor
    |----> panel_simple.c    -- Un exemplu simplu despre paneluri

```

Exista un fisier Makefile de nivel inalt in directorul principal. Acesta construiește toate fisierele și pune fisiere exe gata de folosire in directorul demo/exe. Poti face make selectiv mergand in directorul corespunzator. Fiecare director contine un fisier README care explica scopul fiecarui fisier c din director.

Pentru fiecare exemplu am dat numele cai fisierului relativ la directorul ncurses.

Daca doriti puteti vedea programe individuale. Toate programele sunt create sub si puteti sa le folositi pentru oric doriti.

## Credite

Mulumesc lui [Sharath](#) si lui Emre Akbas pentru ajutorul la cateva sectiuni. Introducerea a fost initial scrisa de [Sharath](#). Am rescris-o pastrand cateva fragmente din lucrarea lui initiala. Emre a ajutat la scrierea sectiunilor printw si scanw.

Apoi urmeaza Ravi Parimi, dragul meu prieten, care a participat la acest proiect inainte ca macar o linie sa fi fost scrisa. El m-a 'bombardat' mereu cu sugestii si a revizuit cu rabdare intregul text. De asemenea el a verificat fiecare program din Linux si Solaris. Vezi notitele lui pentru a-ti verifica problemele.

## Lista de dorinte

Aceasta este lista de dorinte, in ordinea prioritatilor. Daca aveti o dorinta sau vreti sa lucrati la completarea unei dorinte, scrieti-mi.

- Aadauga exemple la ultimele parti din sectiunea formulare. ( Lucrez la asta)
- Pregateste un demo aratand toate programele si permite utilizatorului sa parcurga descrierea fiecarui program. Lasa utilizatorul sa compileze si sa vada programul in actiune. Un dialog bazat pe interfata este preferat. (Prietenul meu N.N.Ashok lucreaza la asta)
- Aadauga debug info. `_tracef`, `_tracemouse`
- Accesarea `termcap`, `terminfo` folosind functii furnizate de pachetul `ncurses`
- Lucrul la doua terminale simultan
- Aadauga lucruri in sectiunea `inselatoare`

## Schimba Log-ul

Versiunea 1.3.2

- Sters toate referintele la site-ul home
- Corectat `Makefiles`

Versiunea 1.3.1

- Schimbat detial in detail
- Corectat paragrafele ramase
- Corectat numerele stabile create

Versiunea 1.3

- Aadaugat notite despre dreptul de copiere (licenta LDP) la documentul principal
- S-a pus de asemenea notite despre dreptul de copiere (GPL) pentru programe
- Corectat exemplul `printw`.

## Versiunea 1.2

- Incorporat schimbarile ravi. In principal in sectiunile introducere, meniui, formular, distractiv
- Cateva schimbari dragute pentru a face sa arate bine in IE.

## Versiunea 1.1

- S-a adaugat sectiunea “ cate ceva despre ferestre”
- S-a corectat o multine de greseli de exprimare
- S-a adugat exemplul scanw.

# Drepturi de copiere

Copyright (c) 2001 de Pradeep Padala. Acest document poate fi distribuit in termenii stabiliti prin licenta LDP la [linuxdoc.org/COPYRIGHT.html](http://linuxdoc.org/COPYRIGHT.html).

Acest HOWTO este documentatie gratis ; puteti sa o redistribuiti si/sau sa o modificati in termenii licentei LDP. Acest document este distribuit cu speranta ca va fi folositor, dar fara nici o garantie; chiar si fara garantia ceruta de mercantibilitate sau fitness pentru un scop particular. Vezi licenta LDP pentru mai multe detalii.

# Programul Hello World

Bine ati venit in lumea curses. Inainte de a intra in librarii si a vedea variatele lor caracteristici, sunete si fluieraturi, sa scriem un simplu program si sa salutam lumea.

## Compilarea cu librariile Ncurses

Pentru a folosi functiile de librarie din ncurses, trebuie sa includeti ncurses.h si sa legati programul cu librariile ncurses trebuind adaugat steguletul -lncurses. Ncurses.h include deja stdio.h

```
#include <ncurses.h>
```

```
·  
·  
·
```

```
compilare si legare: gcc <program file> -lncurses
```

## Codul

```
                                Programul Hello World !!!

/* Cale fisier: basics/hello_world.c */
#include <ncurses.h>

int main()
{
    initscr();                    /* Incepe modul curses          */
    printw("Hello World !!!");   /* Afiseaza Hello World       */
    refresh();                   /* Actuleaza-l pe ecranul real */
    endwin();                    /* Sfarsitul modului curses   */
    return 0;
}
```

## Disecare

Programul de mai sus afiseaza pe ecran “Hello World !!!” si se termina. Acest program arata cum sa initializezi curses si sa folosesti ecranul si sa sfarsesti modul curses. Sa disecam linie cu linie programul.

### Despre initscr()

Funcția `initscr()` initializează terminalul în modul `curses`. În unele implementări, ea șterge ecranul și afișează un ecran gol. Pentru orice administrare a ecranului folosind pachetul `curses` aceasta trebuie apelată prima. Această funcție initializează sistemul `curses` și alocă memorie pentru fereastra curentă numită ‘`stdscr`’ și pentru alte structuri de date. În cazuri extreme această funcție poate esua din cauza memoriei insuficiente pentru alocarea memoriei pentru structurile de date ale bibliotecii `curses`.

După aceasta putem face o varietate de inițializări pentru a ne customiza setările `curses`. Aceste detalii vor fi explicate mai târziu.

### Misteriosul refresh()

Următoarea linie `printw` afișează șirul “Hello World!!!” pe ecran. Această funcție este analog funcției normale `printf` în toate aspectele cu excepția că ea afișează datele într-o fereastră numită `stdscr` la coordonatele curente (y,x). Din moment ce coordonatele noastre curente sunt la 0,0 șirul este afișat în colțul stâng al ferestrei.

Aceasta ne duce apoi la misteriosul `refresh()`. Așadar, când am apelat `printw` datele au fost scrise într-o fereastră imaginară numită `stdscr`, care nu este încă actualizată pe ecran. Rolul lui `printw` este să actualizeze câteva stegulete și structuri de date și să scrie datele într-un buffer

corespunzator lui `sdsr`. Pentru a-l aduce pe ecran trebuie sa apelam `refresh()` si sa transmitem sistemului curses sa lase continutul pe ecran.

Filozofia din spate este sa permita programatorului sa faca multiple actualizari pe ecranul imaginar sau pe ferestre si sa actualizeze o data toate actualizarile ecranului au fost facute. Functia `refresh()` verifica fereasta si actualizeaza doar portiunea care a fost schimbata. Aceasta ofera un rezultat bun si ofera de asemenea o mare flexibilitate. Dar uneori este frustranta pentru incepatori. O greseala des facuta de incepatori este sa uite sa apeleze `refresh()` dupa ce au facut ceva actualizari folosind clasa de functii `printw()`. Inca uit si eu sa o adaug uneori ☺

## Despre `endwin()`

Si in final sa nu uitati sa inchideti modul curses. Altminteri terminalul dumneavoastra ar putea sa se comporte ciudat dupa ce programul se inchide. Functia `endwin()` elibereaza memoria folosita de sub-sistemul curses si structurile sale si aduce terminalul in modul normal. Acesta functie trebuie sa fie apelata dupa ce ai terminat cu modul curses.

# Detaliile sangeroase

Acum ca am vazut cum sa scriem un simplu program in curses sa intram in detalii. Exista multe functii care ajuta la customizarea a ceea ce vedem pe ecran si multe caracteristici care pot fi folosite pe deplin.

Sa incepem...

## Initializarea

Stim ca pentru a initializa sistemul curses trebuie apelata functia `initscr()`. Exista functii care pot fi apelate dupa aceasta initializare pentru a ne customiza sesiunea curses. Putem cere sistemului curses sa seteze terminalul in modul raw sau sa initializeze cuoarea sau sa initializeze mouse-ul etc.. Sa discutam despre cateva functii care sunt apelate in mod normal dupa `initscr()`;

## Sa discutam despre functii de initializare ca `raw()` etc..

**`raw()` Si `cbreak()`**

In mod normal administratorul terminalului pastreaza caracterele pe care un utilizator le tasteaza pana cand se intalneste o linie noua sau este apasat Enter. Dar cele mai multe programe cer sa fie caracterele disponibile imediat ce utilizatorul le tasteaza. Functiile de mai sus sunt folosite sa stopeze pastrarea liniei. Diferenta dintre aceste doua functii consta in felul cum caracterele de control cum ar fi suspendarea (CTRL-Z), intrerupere si terminare (CTRL-C) sunt trimise programului. In modul `raw()` aceste caractere sunt transmise direct programului

fara a genera un semnal. In modul cbreak() aceste caractere de control sunt interpretate ca orice alte caractere de administratorul terminalului. Eu personal prefer folosirea lui raw() pentru ca pot exercita mai mult control asupra ceea ce face utilizatorul.

### **echo() Si noecho()**

Aceste functii controleaza rasunetul pe care caracterele tastate de utilizator catre terminal. Noecho() schimba in inchis ecoul. Motivul pentru care ai vrea sa faci asta este ca sa castigi mai mult control asupra ecoului sau pentru a inabusi prea mult ecou in timp ce primești intrari de la utilizator prin functiile getch() etc. Multe dintre programele iterative apeleaza noecho() la initializare si fac transmiterea caracterelor intr-o maniera controlata. Aceasta da programatorului flexibilitatea transmiterii caracterelor in orice loc pe fereastra fara actualizarea coordonatelor (y,x) curente.

### **keypad()**

Aceasta este functia mea de initializare favorita. Ea da posibilitatea citirii functiilor taste ca F1, F2, taste sageti etc. Aproape orice program iterativ da acesta posibilitate, din cauza ca tastele sageti sunt o parte importanta a oricarei Interfete cu Utilizatorul. Apelati keypad(stdscr, TRUE) pentru a porni acesta caracteristica pentru ecranele obisnuite. Vetii afla mai multe despre folosirea tastelor in sectiunile urmatoare ale acestui document.

### **halfdelay()**

Aceasta functie, desi folosita destul de rar, este folositoare uneori. Functia halfdelay() este apelata pentru a porni modul half-delay, care este similar cu modul cbreak() prin aceea ca caracterele tastate sunt imediat disponibile programului. Oricum, asteapta 'X' zeci de secunda intrarea si returneaza ERR, daca nici o intrare nu este disponibila. 'X' este timpul limita valoare transmisa functiei halfdelay(). Aceasta functie este folositoare cand vreti sa cereti utilizatorului o intrare, si daca el nu raspunde intr-un anumit timp, putem face altceva. Un exemplu posibil este un timp limita la cerearea parolei.

## **Variate functii de initializare**

Mai exista cateva functii care sunt apelate la initializare pentru a customiza comportarea curses. Ele nu sunt folosite atat de des ca si cele mentionate mai sus. Cateva dintre ele sunt explicate la momentul potrivit.

## **Un exemplu**

Sa scriem un program care va clarifica utilizarea acestor functii.

### **Program exemplu de functii de initializare**

```
/* Cale fisier: basics/init_func_example.c */
#include <ncurses.h>
```

```

int main()
{
    int ch;

    initscr();                /* Incepe modul curses          */
    raw();                    /* Stoparea pastrarii liniei  */
    keypad(stdscr, TRUE);    /* Luam F1, F2 etc..         */
    noecho();                 /* Nu apela echo() in timp ce apelam getch */

    ch = getch();            /* daca raw() nu ar fi apelata
                             * trebuie sa apasam enter inaintea ca
ea*/

    if(ch == KEY_F(1))      /* sa ajunga la program      */
    ajunge la noi*/        /* Fara keypad deschis aceasta nu ar
                             * ajunge la noi*/
        printf("F1 Key pressed");

    else                    /* Fara noecho() cateva caractere*/
    {                       /* neplacute ar fi afisate pe ecran*/
        printf("The pressed key is ");
        attron(A_BOLD);
        printf("%c", ch);
        attroff(A_BOLD);
    }

    refresh();              /* Afiseaza pe ecranul real */
    endwin();               /* inchde modul curses      */

    return 0;
}

```

Acest program se auto explica. Dar am folosit functii pe care nu le-am explicat inca. Functia getch() este folosita pentru a citi un caracter de la utilizator. Este echivalenta cu getchar() cu exceptia ca putem stopa pastrarea liniei pentru a evita <enter> dupa intrare. Vedeti mai multe despre getch() si despre citirea tastelor in sectiunea administrarea tastelor. Functiile attron si attroff sunt folosite pentru a seta anumite atribute , respectiv a le scoate. In exemplu le-am folosit pentru a afisa caracterele bold. Aceste functii sunt explicate in detaliu mai tarziu.

## Cateva cuvinte despre Ferestre

Inainte de a patrunde in nenumaratele functii ncurses, sa clarific cateva lucruri despre ferestre. Ferestrele sunt explicate in detaliu in urmatoarele sectiuni.

O fereastră este un ecran imaginar definit de sistemul curses. O fereastră nu însemna o fereastră cu margini întalnită de obicei pe platformele Win9X. Când curses este initializat, acesta creează o fereastră implicită numită stdscr, care reprezintă ecranul vostru de 80x25( sau dimensiunea ferestrei în care rulați). Dacă realizați lucruri simple ca afisarea de siruri putine, citirea intrării etc., puteti folosi în siguranța aceasta singura fereastră pentru toate scopurile voastre. Puteti crea de asemenea ferestre și apela functii care lucrează în mod explicit la fereastră specificată.

De exemplu, dacă apelezi

```
printw("Hi There !!!");
refresh();
```

Afiseaza sirul pe stdscr la pozitia curenta a cursorului. In mod similar alelul refresh(), lucreaza doar cu stdscr.

Sa spunem ca ai creat ferestre apoi trebuie sa aplelezi o functie cu 'w' adaugat la numele functiei obisnuite.

```
wprintw(win, "Hi There !!!");
wrefresh(win);
```

Dupa cum veti vedea in retul documentului, numirea functiilor urmeaza aceeasi conventie. Pentru o functie mai exista de obicei inca alte 3 functii.

```
printw(string);          /* Afiseaza la stdscr la pozitia curenta a
cursorului*/
mvprintw(y, x, string); /* Se muta la (y,x) apoi afiseaza sirul*/
printw(win, string);    /* Afiseaza pe fereastra win la pozitia
curenta a cursorului*/          /* in the window */
mvwprintw(win, y, x, string); /* Se muta la (y, x) relativ la
coordonatele ferestrei apoi afiseaza*/
```

De obicei functiile fara w sunt macrouri care se extind la functiile cu w corespunzatoare cu stdscr ca parametru al ferestrei.

## Despre functii de iesire ca printw()

Presupun ca de abia asteptati sa vedeti ceva actiune. Inapoi la odissea despre functiile curses. Acum ca curses este initializat, sa interactionam cu lumea.

Acestea sunt trei clase de functii pe care le poti folosi pentru iesirea pe ecran.

1. clasa addch() : Afiseaza un singur caracter cu atributele sale
2. clasa printw() : Afiseaza iesire formatata ca si printf()
3. clasa addstr() : Afiseaza siruri

Aceste functii pot fi inteschimbate si e doar o problema de stil ce clasa sa folosesti. Sa vedem fiecare in detaliu.

### Clasa de functii addch()

Aceste functii pun un singur caracter la locatia curenta a cursorului si avanseaza pozitia cursorului. Poti da un caracter sa fie afisat dar de obicei acestea sunt folosite pentru a afisa un caracter cu anumite atribute. Atributele sunt explicate in detaliu in sectiunile ulterioare ale documentului. Daca un caracter este asociat cu un atribut( bold, reverse video etc.), cand curses afiseaza un caracter, acesta este afisat cu acel atribut.

Pentru a combina un caracter cu anumite atribute, aveti doua optiuni:

- Folosind OR la un singur caracter cu macrourele de atribute dorite. Aceste macroure de atribute pot fi gasite in header-ul ncurses.h. De exemplu, Vreti sa afisati un caracter ch(de tipul char) bold sau subliniat, veti apela addch() ca mai jos.
- ```
addch(ch | A_BOLD | A_UNDERLINE);
```
- Folosint functii ca attrset(), attron(), attroff(). Aceste functii sunt explicate in sectiunea atribute. Pe scurt, ele manipuleaza atributele curente ale ferestrei date. O data setat, caracterul afisat pe fereastra este asociat cu atributele pana cand acestea sunt oprite.

Ca adaugare, curses furnizeaza cateva caractere speciale pentru caractere bazate pe grafica. Poti desena tabele, linii horizontale sau verticale, etc. Poti gasi toate caracterele disponibile in fisierul header ncurses.h. Incearca sa gasesti macrourele incepand cu ACS\_ in acest fisier.

### **mvaddch(), waddch() SI mvwaddch()**

mvaddch() este folosita pentru a muta cursorul la un punct dat, si apoi afisa. Asadar, apelurile:

```
move(row,col); /* muta cursorul la linia row si coloana col*/  
addch(ch);
```

pot fi inlocuite cu :

```
mvaddch(row,col,ch);
```

waddch() este similar cu addch(), cu exceptia ca acesta adauga un caracter la fereastra data. (Retine ca addch() adauga un caracter la fereastra stdscr.)

Intr-un mod similar mvaddch() este folosita pentru a aduga un caracter in fereastra data la coordonatele date.

Acum, suntem familiari cu functiile elementare de iesire addch(). Dar, daca vrem sa afisam un sir de caractere, ar fi foarte suparator sa il afisam caracter cu caracter. Din fericire, ncurses furnizeaza functii la fel ca printf si put.

### **Clasa de functii printf()**

Aceste functii sunt similare cu printf() in plus cu capacitatea afisarii la orice pozitie pe ecran.

### **printf() SI mvprintf**

Aceste doua functii lucreaza aproape la fel ca printf(). Mvprintf() poate fi folosit pentru a muta cursorul la o pozitie si apoi afisa. Daca vrei sa muti cursorul mai inati si apoi afisa folosind functia printf(), foloseste move() mai inati si apoi foloseste printf() desi nu vad de ce cineva ar evita sa foloseasca mvprintf(), ai posibilitatea de manipulare.

## **wprintw() și mvprintw**

Aceste doua functii sunt similare cu cele doua de sus cu exceptia ca ele afiseaza n fereastra corespunzatoare data ca argument.

## **vwprintw()**

Aceasta functie este similara cu `vprintf()`. Aceasta poate fi folosita cand un numar variabil de argumente trebuie afisate.

## **Some examples**

```
/* cale fisier: basics/printw_example.c */
#include <ncurses.h>                /* ncurses.h include stdio.h */
#include <string.h>

int main()
{
    char mesg[]="Just a string";      /* mesaj aparut pe ecran*/
    int row,col;                      /* pentru a pastra numarul de coloane
si linii */
    initscr();                        /* incepe modul curses */
    getmaxyx(stdscr,row,col);         /* afla numarul de linii si coloane */
    mvprintw(row/2,(col-strlen(mesg))/2,"%s",mesg);
  /* afiseaza mesajul in centrul
ecranului */
    mvprintw(row-2,0,"This screen has %d rows and %d columns\n",row,col);
    printw("Try resizing your window(if possible) and then run this program
again");
    refresh();
    getch();
    endwin();

    return 0;
}
```

Programul de mai sus demonstreaza cat de simplu este sa folosesti `printw`. Dai doar coordonatele si mesajul va aparea pe ecran, apoi face ceea ce doriti.

Programul de mai sus ne introduce spre o noua functie `getmaxyx()`, un macro definit in `ncurses.h`. Acesta da numarul de de coloane si numarul de linii intr-o fereastra data. `Getmaxyx()` face asta prin actualizarea variabilelor date ei. Din moment ce `getmaxyx()` nu este o functie, nu transmitem pointeri ei, dam doar doua variabile intregi.

## Clasa de functii `addstr()`

`addstr()` este folosita pentru a pune un sir de caractere intr-o feresatra data. Aceasta functie este similara cu apelul `addch()` o data pentru fiecare caracterdintr-un sir de caractere dat. Aceasta este la fel pentru toate functiile de iesire. Sunt alte functii din aceasta familie ca `mvaddstr()`, `mvwaddstr()` si `waddstr()`, care folosesc conventia de nume din curses.(e.g. `mvaddstr()` este similara cu apelul `move()` si apoi `addstr()`). Alat functie din aceasta familie este `addnstr()`, care ia in plus un intreg(sa spunem `n`) ca parametru. Aceasta functie pune cel mult `n` caractere pe ecran. Daca `n` este negativ, atunci tot sirul va fi adaugat.

## Precutie

Toate aceste functii iau mai intai coardaonata `y` si apoi `x` in argumentele lor. O greseala obisnuita la incepatori este sa paseze `x,y` in aceasta ordine. Daca faci prea multe manipulari ale coordonatelor (`y,x`), gandeste-te la impartirea ecranului in ferestre si sa manipulezi fiecare separat.Ferestrele sunt explicate in sectiunea ferestre.

## Despre functii de intrare ca `scanw()`

Afisarea fara sa iei intrari este plictisitoare. Sa vedem functii care ne permit sa luam intrari de la utilizator. Aceste functii pot fi de asemenea impartite in trei categorii.

1. clasa `getch()`: Ia un caracter
2. clasa `scanw()` : Ia intrare formatata
3. clasa `getstr()` : Ia siruri de caractere

Clasa de functii `getch()`

Aceste functii citesc un singur caracter de la terminal. Dar exista mai multe lucruri subtile de considerat. De exemplu daca nu folosesti functia `cbreak()`, `curses` nu va citi caracterele din intrare in continuare, ci vor incepe citirea lor numai dupa o linie noua sau dupa ce este intalnit EOF. Pentru a evita asta, functia `cbreak()` trebuie folosita astfel incat caracterele sa fie imediat disponibile programului tau. O alta functie des folosita este `noecho()`. Dupa cum numele sugereaza, cand aceasta functie este folosita, caracterele care sunt tastate de utilizator nu vor aparea pe ecran. Cele doua functii `cbreak()` si `noecho()` sunt exemple tipice de administrare a tastelor. Functii de acest gen sunt explicate in sectiunea administrarea tastelor.

## Clasa de functii `scanw()`

Aceste functii sunt similare cu `scanf()` in plus cu capacitatea de a lua intrarea de la orice locatie de pe ecran.

## **scanw() și mvscanw**

Utilizarea acestor functii este similara cu sscanf(), unde linia care trebuie scanata este data de functia wgetstr(). De aceea, aceste functii apeleaza wgetstr()(explicata mai jos) si folosesc linia rezultata pentru o scanare.

## **wscanw() și mvwscanw()**

Acestea sunt similare cu cele doua functii de mai sus cu exceptia ca ele citesc de la o fereastră, care este data ca argument acestor functii.

## **vwscanw()**

Aceasta functie este similara cu vscanf(). Aceasta poate fi folosita cand un numar variabil de argumente trebuie scanate.

## **Clasa de functii getstr()**

Aceste functii sunt folosite sa ia siruri de caractere de la terminal. In esenta, aceasta functie face acelasi lucru care ar fi fost obtinut print-o serie de apeluti getch() pana la o linie noua, enter, sau pana la sfarsit de fisier. Sirul de caractere rezulatat este reprezentat de str, care este un pointer de caractere furnizat de utilizator.

## **Cateva exemple**

```
/* Cale fisier: basics/scanw_example.c */
#include <ncurses.h> /* ncurses.h include stdio.h */
#include <string.h>

int main()
{
    char mesg[]="Enter a string: "; /* mesaj care trebuie sa apara
pe ecran*/
    char str[80];
    int row,col; /* pentru a stoca numarul de linii*/
    initscr(); /* incepe modul curses */
    getmaxyx(stdscr,row,col); /* obtine numarul de linii si
coloane*/
    mvprintw(row/2,(col-strlen(mesg))/2,"%s",mesg);
/* afiseaza mesajul in centrul ecranului*/
    getstr(str);
    mvprintw(23, 0, "You Entered: %s", str);
    getch();
    endwin();

    return 0;
}
```

# Atribute

Am vazut un exemplu cum pot fi atributele folosite pentru a afisa caractere cu unele efecte speciale. Atributele, cand sunt setate prudent, pot prezenta informatii intr-o maniera usoara si inteligibila. Urmatorul program ia un fisier C ca intrare si afiseaza fisierul cu comentarii ingrosate. Uitati-va prin cod.

## Exemplu

```
/* File path: basics/simple_attr.c */
#include <ncurses.h>

int main(int argc, char *argv[])
{
    int ch, prev;
    FILE *fp;
    int goto_prev = FALSE, y, x;

    if(argc != 2)
    {
        printf("Usage: %s <a c file name>\n", argv[0]);
        exit(1);
    }
    fp = fopen(argv[1], "r");
    if(fp == NULL)
    {
        perror("Cannot open input file");
        exit(1);
    }

    initscr();                                /* Incepe modul curses          */

    prev = EOF;
    while((ch = fgetc(fp)) != EOF)
    {
        if(prev == '/' && ch == '*')          /* Daca este / sau * atunci
  * schimbati in bold */
        {
            attron(A_BOLD);
            goto_prev = TRUE;                 /* Merge la anteriorul char /
si *si afiseaza-l in BOLD
  */
        }
        if(goto_prev == TRUE)
        {
            getyx(stdscr, y, x);
            move(y, x - 1);
            printw("%c%c", '/', ch); /* Afisarea actuala este
facuta*/
  * aici*/
            goto_prev = FALSE;               /* Seteaza-l pe FALSE sau
totul de aici incolo va fi / */
        }
        else
            printw("%c", ch);
        refresh();
        if(prev == '*' && ch == '/')
    }
}
```

```

        attroff(A_BOLD);          /* Schimba-l o data ce avem *
si apoi */
        prev = ch;
    }

    endwin();                    /* Inchide modul curses */

    return 0;
}

```

Nu va faceti griji in legatura cu toate acele initializari si alte chestii. Concentrati-va asupra buclei while. Ea citeste fiecare caracter din fisier si cauta paternul /\*. O data gasit paternul, seteaza atributul BOLD cu attron(). Cand gasim paternul \*/ este oprit cu attroff().

Programul de mai sus ne introduce doua functii folositoare getyx() si move(). Prima functie obtine coordonatele curente ale cursorului in variabilele y, x. Din moment ce getyx() este un macro nu trebuie sa pasam pointeri variabilelor. Functia move() muta cursorul la coordonatele date de ei.

Programul de mai sus este unul simplu care nu face multe. In locul acestor linii se poate scrie un program mai folositor care citeste un fisier C, il parseaza si il afiseaza in diferite culori. Se poate chiar extinde in alte limbaje de asemenea.

## Detaliile

Sa intram in mai multe detalii despre atribute. Functiile attron(), attroff(), attrset() , si functiile lor inrudite attr\_get() etc..pot si folosite pentru a comuta atributele in pornit/oprit, pentru a obtine atributele si a produce un afisaj colorat.

Functiile attron si attroff iau o masca de biti ale atributelor si le comuta in pornit, respectiv oprit. Urmatoarele atribute video, care sunt definite in <curses.h> pot fi trimse acestor functii.

|               |                                              |
|---------------|----------------------------------------------|
| A_NORMAL      | Afisaj normal(fara iluminare)                |
| A_STANDOUT    | Cel mai bun mod de iluminare a terminalului  |
| A_UNDERLINE   | Subliniere                                   |
| A_REVERSE     | Video reverse                                |
| A_BLINK       | Licarire                                     |
| A_DIM         | Jumatate luminos                             |
| A_BOLD        | Foarte luminos sau ingrosat                  |
| A_PROTECT     | Modul protejat                               |
| A_INVIS       | Modul invizibil sau gol                      |
| A_ALTCHARSET  | Alterneaza setul de caractere                |
| A_CHARTEXT    | Masca de biti pentru extragerea caracterelor |
| COLOR_PAIR(n) | Perechea de culori numarul n                 |

Ultimul este cel mai colorat☺. Culorile sunt explicate in sectiunile urmatoare.

Putem folosi oricate atribute de mai sus folosind OR() pentru a obtine efecte combinate. Daca vreti video reverse impreuna cu caractere licarind putei folosi

```
attron(A_REVERSE | A_BLINK);
```

### **attron() VS attrset()**

Atunci care este diferenta dintre attron() si attrset()? attrset seteaza atributele ferestrei pe cand attron doar comuta in pornit atributul dat ei. Deci attrset() suprascrie in intregime orice atribut pe care fereastra le-a avut inainte si ii seteaza noile atribute. La fel, attron() doar comuta in inchis toate atributele date ei ca argument. Aceasta ne da posibilitatea de a lucra usor cu atributele. Dar daca le folositi neatent putei pierde sirul atributelor pe care fereastra le are si sa gresiti afisarea. Aceasta se adevereste cand lucram cu meniuri cu culori si iluminare. Asadar decideti asupra unei politici consistente si urmati-o. Putei folosi intotdeauna standerd() care este echivalent cu attrset(A\_NORMAL) care opreste toate atributele si va readuce la modul normal.

### **attr\_get()**

Funcția attr\_get() prreai atributele si perechea de culori curente ale ferestrei. Desi noi probabil nu vom folosi aceasta la fel de des ca si functiile de mai sus, aceasta este folositoare in scanarea unor arii din ecran. Sa spunem ca vrem sa facem niste actualizari complexe ale ecranului si nu suntem siguri cu ce atribut este asociat fiecare caracter. Atunci aceasta functie poate fi folosita impreuna cu attrset sau attron pentru a produce efectul dorit.

## **Funcțiile attr\_**

Exista serii de functii ca attr\_set(), attr\_on etc.. Acestea sunt similare cu functiile de mai sus cu exceptia ca ele primesc parametru de tip attr\_t.

## **Funcțiile wattr**

Pentru fiecare din functiile de mai sus exista functia corespunzatoare cu 'w' care opereaza asupra unei ferestre particulare. Functiile de mai sus opereaza asupra lui stdscr.

## **Funcțiile chgat()**

Funcția chgat() este listata la sfarsitul paginei principale curs\_attr. Este de fapt o functie folositoare. Aceasta functie este folosita pentru a seta atribute pentru grupuri de caractere fara mutare. Ma refer fara mutarea cursorului☺. Ea schimba atributele pentru un numar dat de caractere incepand de la pozitia curenta a cursorului.

Putem transmite -1 ca numar de caractere pentru o actualizare pana la sfarsitul liniei. Daca doriti sa schimbati atributele caracterelor de la pozitia curenta pana la sfarsitul liniei, folositi astfel:

```
chgat(-1, A_REVERSE, 0, NULL);
```

Acesta functie este utila atunci cand doriti sa schimbati atributele caracterelor atunci cand ele sunt deja afisate pe ecran. Mutati-va la caracterul incepand de unde vreti sa schimbati si schimbati atributul.

Alte functii `wchgat()`, `mvchgat()`, `wchgat()` se comporta la fel cu exceptia ca functiile 'w' opereaza asupra unei ferestre particulare. Functiile `mv` muta mai intai cursorul si apoi fac ceea ce au de facut. De fapt `chgat` este un macro care este inlocuit cu `wchgat()` avand ca fereastră `stdscr`. Multe dintre functiile fara 'w' sunt macrouri.

## Exemplul `chgat()`

```
/* File path: basics/with_chgat.c */
#include <ncurses.h>

int main(int argc, char *argv[])
{
    initscr();
    start_color();
    init_pair(1, COLOR_CYAN, COLOR_BLACK);
   printw("A Big string which i didn't care to type fully ");
    mvchgat(0, 0, -1, A_BLINK, 1, NULL);
    /* Primii doi parametri specifica pozitia la care sa inceapa
    *Al treile parametru reprezinta numarul de caractere de actualizat.
    -1 inseamna pana la sfarsitul liniei*/

    /* Al patrulea parametru este atributul normal pe care vrei sa il
    dai caracterului*/
    /* Al cincilea este indexul culorii. Este indexul dat prin
    init_pair()*/
    /* folositi 0 daca nu vreti culoare.
    * Al saselea este intotdeauna NULL
    */
    refresh();
    endwin();
    return 0;
}
```

Acest exemplu ne introduce in lumea culorilor din `curses`. Culorile vor fi explicate in detaliu mai tarziu. Folositi 0 daca nu vreti culoare.

## Totul despre functiile de ferestre

Ferestrele formaaza cel mai important concept in `curses`. Ati vazut fereastra standard `stdscr` asupra careia opereaza toate functiile in mod implicit. Acum pentru a pentru a construi designul unei simple GUI, trebuie sa apelati la ferestre. Motivul principal pentru care ati dori sa folositi ferestre este ca sa puteti manipula parti ale ecranului separat, pentru o mai mare eficacitate, prin actualizarea doar a ferestrelor pe care doriti sa le modificati pentru o mai buna proiectare. As spune ca ultimul motiv este cel mai puternic pentru a folosi ferestre. Ar trebui ca intotdeauna sa cautati o proiectare mai buna si mai usor de manevrat in programele voastre.

Daca aveti de scris un GUI mare si complex aceasta este de cea mai mare importanta inainte de a incepe altceva sa faceti.

## Bazele

O fereastră poate fi creată prin apelul funcției `newwin()`. Ea nu creează de fapt nimic pe ecran. Ea alocă memorie pentru o structură pentru a manipula fereastra și actualizează structura cu date privind fereastra cum ar fi dimensiunea, `beginy`, `beginx` etc.. De acum încolo în `curses`, o fereastră este doar o abstracție a unei ferestre imaginare, care poate fi manipulată independent de celelalte părți ale ecranului. Funcția `newwin()` returnează un pointer la structura `WINDOW`, care poate fi transmisă la funcții legate de ferestre cum ar fi `wprintw()` etc.. În final fereastra poate fi distrusă cu `delwin()`. Va desaloca memoria asociată cu structura ferestrei.

## Sa fie o fereastră!!!

Cât de distractiv este dacă o fereastră este creată și o putem vedea. Partea frumoasă începe prin afișarea ferestrei. Funcția `box()` poate fi folosită pentru a desena margini în jurul ferestrei. Să explorăm aceste funcții în detaliu în acest exemplu.

## Exemplu Border

```
/* File path: basics/win_border.c */
#include <ncurses.h>

WINDOW *create_newwin(int height, int width, int starty, int startx);
void destroy_win(WINDOW *local_win);

int main(int argc, char *argv[])
{
    WINDOW *my_win;
    int startx, starty, width, height;
    int ch;

    initscr();                /* Incepe modul curses */
    cbreak();                /* Pastrarea liniei in buffer este
dezactivata, pasati totul mie*/
    keypad(stdscr, TRUE);    /* Am nevoie de F1 */

    height = 3;
    width = 10;
    starty = (LINES - height) / 2; /* Calculez pentru o plasare centrala
a ferestrei*/
    startx = (COLS - width) / 2;
    printw("Press F1 to exit");
    refresh();
    my_win = create_newwin(height, width, starty, startx);

    while((ch = getch()) != KEY_F(1))
    {
        switch(ch)
        {
            case KEY_LEFT:
                destroy_win(my_win);
                my_win = create_newwin(height, width,
```

```

starty,--startx);
        break;
        case KEY_RIGHT:
            destroy_win(my_win);
            my_win = create_newwin(height, width,
starty,++startx);
        break;
        case KEY_UP:
            destroy_win(my_win);
            my_win = create_newwin(height, width, --
starty,startx);
        break;
        case KEY_DOWN:
            destroy_win(my_win);
            my_win = create_newwin(height, width,
++starty,startx);
        break;
    }
}

endwin(); /* Sfarsit de mod curses */
return 0;
}

WINDOW *create_newwin(int height, int width, int starty, int startx)
{
    WINDOW *local_win;

    local_win = newwin(height, width, starty, startx);
    box(local_win, 0 , 0); /* 0, 0 da caracterele implicite
        * pentru liniile verticale
        * si orizontale
    */
    wrefresh(local_win); /* Arata boxa */

    return local_win;
}

void destroy_win(WINDOW *local_win)
{
    /* box(local_win, ' ', ' '); : aceasta nu va da rezultatul dorit
    * de stergere a ferestrei. Va lasa cele 4 colturi ale sale
    * precum si un rest al ferestrei neplacut */
    wborder(local_win, ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ');
    /* Parametri luati sunt
    * 1. win: fereastra pe care sa operam
    * 2. ls: caracterul pentru partea stanga a ferestrei
    * 3. rs: caracterul pentru partea dreapta a ferestrei
    * 4. ts: caracterul pentru partea de sus a ferestrei
    * 5. bs: caracterul pentru partea de jos a ferestrei
    * 6. tl: caracterul pentru coltul stanga sus al ferestrei
    * 7. tr: caracterul pentru coltul dreapta sus a ferestrei
    * 8. bl: caracterul pentru coltul stanga jos al ferestrei
    * 9. br: caracterul pentru coltul dreapta jos al ferestrei */
    wrefresh(local_win);
    delwin(local_win);
}

```

## Explicatie

Nu tipa! Stiu ca este un exemplu mare. Dar trebuie sa explic cateva lucruri importante aici☺.Acest program creaza o fereastra dreptunghiulara care poate fi mutata cu sagetile stanga, dreapta, sus, jos. El creaza si distruge in mod repetat ferestre atunci cand utilizatorul apasa o tasta. Nu mergeti peste limitele ecranului. Verificarea acestor limite este lasata ca exercitiu pentru cititori. Sa il disecam linie cu linie.

Functia `create_newwin()` creaza o fereastră cu `newwin()` și afiseaza o margine în jurul ei. Functia `destroy_win()` mai întâi șterge fereastra de pe ecran desenând o margine cu caracterul ‘’ și apoi apelează `delwin()` pentru a desaloca memoria legată de ea. În funcție de tasta pe care utilizatorul o apasă, stary sau startx se schimbă și o nouă fereastră este creată.

În `destroy_win`, după cum vedeți, am folosit `wborder` în loc de `box`. Motivul este scris în comentarii( Ai ratat asta.Stiu. Citeste codul☺). `Wborder` desenează o margine în jurul ferestrei cu caracterele date ei ca fiind cele patru puncte ale colturilor și cele patru linii. Pentru lamurire, dacă ați fi apelat `wborder` ca mai jos :

```
wborder(win, '|', '|', '-', '-', '+', '+', '+', '+');
```

se produce ceva de genul

```
+-----+
|       |
|       |
|       |
|       |
+-----+
```

## Celelalte lucruri din exemplu

Se pot vedea în exemplul de mai sus, că am folosit variabilele `COLS,LINES` care sunt inițializate la dimensiunile ferestrei după `initscr()`. Ele pot fi utile în determinarea dimensiunilor ferestrei și găsirea coordonatelor centrului ecranului de mai sus. Functia `getch()` ca de obicei preia tasta de la tastatură și în funcție de tasta, ea face treaba corespunzătoare. Acest tip de `switch-case` este foarte des în programele bazate pe GUI.

## Alte functii Border

Programul de mai sus este foarte ineficient din cauza că la fiecare tastare a unei taste, o fereastră este distrusă și alta este creată. Deci să scriem un program mai eficient care folosește alte funcții cu margini înrudite.

Următorul program folosește `mvhline()` și `mvvline()` pentru a obține același efect. Aceste 2 funcții sunt simple. Ele creează o linie orizontală sau verticală de lungime specificată și la o poziție specificată.

## Codul

```
/* File path: basics/other_border.c */
#include <ncurses.h>

typedef struct _win_border_struct {
    chtype ls, rs, ts, bs,
           tl, tr, bl, br;
}WIN_BORDER;

typedef struct _WIN_struct {

    int startx, starty;
    int height, width;
    WIN_BORDER border;
}WIN;

void init_win_params(WIN *p_win);
void print_win_params(WIN *p_win);
void create_box(WIN *win, int bool);

int main(int argc, char *argv[])
{
    WIN win;
    int ch;

    initscr();                /* Incepe modul curses */
    cbreak();                 /* Pastrrea liniei in buffer este
dezactivata, pasati totul mie*/
    keypad(stdscr, TRUE);     /* Am nevoie de F1 */
    noecho();
    init_pair(1, COLOR_CYAN, COLOR_BLACK);

    /* Initialize the window parameters */
    init_win_params(&win);
    print_win_params(&win);

    attron(COLOR_PAIR(1));
    printw("Press F1 to exit");
    refresh();
    attroff(COLOR_PAIR(1));

    create_box(&win, TRUE);
    while((ch = getch()) != KEY_F(1))
    {
        switch(ch)
        {
            case KEY_LEFT:
                create_box(&win, FALSE);
                --win.startx;
                create_box(&win, TRUE);
                break;
            case KEY_RIGHT:
                create_box(&win, FALSE);
                ++win.startx;
                create_box(&win, TRUE);
                break;
            case KEY_UP:
                create_box(&win, FALSE);
```

```

        --win.starty;
        create_box(&win, TRUE);
        break;
    case KEY_DOWN:
        create_box(&win, FALSE);
        ++win.starty;
        create_box(&win, TRUE);
        break;
    }
}
endwin(); /* Sfarsit mod curses */
return 0;
}

void init_win_params(WIN *p_win)
{
    p_win->height = 3;
    p_win->width = 10;
    p_win->starty = (LINES - p_win->height)/2;
    p_win->startx = (COLS - p_win->width)/2;

    p_win->border.ls = '|';
    p_win->border.rs = '|';
    p_win->border.ts = '-';
    p_win->border.bs = '-';
    p_win->border.tl = '+';
    p_win->border.tr = '+';
    p_win->border.bl = '+';
    p_win->border.br = '+';
}

void print_win_params(WIN *p_win)
{
#ifdef _DEBUG
    mvprintw(25, 0, "%d %d %d %d", p_win->startx, p_win->starty,
            p_win->width, p_win->height);

    refresh();
#endif
}

void create_box(WIN *p_win, int bool)
{
    int i, j;
    int x, y, w, h;

    x = p_win->startx;
    y = p_win->starty;
    w = p_win->width;
    h = p_win->height;

    if(bool == TRUE)
    {
        mvaddch(y, x, p_win->border.tl);
        mvaddch(y, x + w, p_win->border.tr);
        mvaddch(y + h, x, p_win->border.bl);
        mvaddch(y + h, x + w, p_win->border.br);
        mvhline(y, x + 1, p_win->border.ts, w - 1);
        mvhline(y + h, x + 1, p_win->border.bs, w - 1);
        mvvline(y + 1, x, p_win->border.ls, h - 1);
        mvvline(y + 1, x + w, p_win->border.rs, h - 1);
    }
}

```

```

    }
    else
        for(j = y; j <= y + h; ++j)
            for(i = x; i <= x + w; ++i)
                mvaddch(j, i, ' ');

    refresh();
}

```

## Totul despre culori

Viata pare neinteresanta fara culori. Curses are un mecanism dragut pentru a lucra cu culori. Sa intram in desimea lucrurilor cu un program simplu.

### Un exemplu simplu

```

/* File path: basics/simple_color.c */
#include <ncurses.h>

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string);
int main(int argc, char *argv[])
{
    initscr();
    /* Incepe modul curses */
    if(has_colors() == FALSE)
    {
        endwin();
        printf("You terminal does not support color\n");
        exit(1);
    }
    start_color();
    /* Incepe culoarea */
    init_pair(1, COLOR_RED, COLOR_BLACK);

    attron(COLOR_PAIR(1));
    print_in_middle(stdscr, LINES / 2, 0, 0, "Viola !!! In color ...");
    attroff(COLOR_PAIR(1));
    endwin();
}

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string)
{
    int length, x, y;
    float temp;

    if(win == NULL)
        win = stdscr;
    getyx(win, y, x);
    if(startx != 0)
        x = startx;
    if(starty != 0)
        y = starty;
    if(width == 0)
        width = 80;
}

```

```

length = strlen(string);
temp = (width - length) / 2;
x = startx + (int)temp;
mvwprintw(win, y, x, "%s", string);
refresh();
}

```

Dupa cum puteti vedea, pentru a incepe folosirea culorilor, trebuie sa apelati mai intai functia `start_color()`. Apoi putei folosi capacitatea de culoare ale terminalului vostru cu diferite functii. Pentru a vedea daca terminalul are capacitati de culoare sau nu, folositi functia `has_colors()`, care returneaza `FALSE` daca terminalul nu suporta culori.

`Curses` initializeaza toate culorile suportate de terminal cand este apelat `start_color()`. Acestea pot fi accesate de constantele definite ca `COLOR_BLACK` etc. Acum pentru a incepe efectiv folosirea culorilor, trebuie sa definiti perechi. Culorile sunt intotdeauna folosite in perechi. Asta inseamna ca trebuie sa folositi functia `init_pair()` pentru a defini prim-planul si fundalul pentru numarul perechii date. Dupa aceea numarul perechii poate fi folosit ca un atribut normal cu functia `COLOR_PAIR()`. Poate parea dificil la inceput. Dar aceasta solutie eleganta ne permite sa folosim perechi de culori foarte usor. Pentru a observa asta, trebuie sa cautati in codul sursa a dialogului, o utilitate pentru afisarea casutelor de dialog din scripturi shell. Programatorii au definit combinatii de prim-plan si fundal pentru toate culorile de care ar avea nevoie si le-au initializat de la inceput. Aceasta usureaza setarea atributelor, prin accesarea unei perechi pe care deja am definit-o ca o constanta.

Urmatoarele culori sunt definite in `curses.h`. Le poti folosi ca parametru pentru diferite functii de culoare.

```

COLOR_BLACK    0
COLOR_RED      1
COLOR_GREEN    2
COLOR_YELLOW   3
COLOR_BLUE     4
COLOR_MAGENTA  5
COLOR_CYAN     6
COLOR_WHITE    7

```

## Schimbarea definitiei unei culori

Functia `init_color()` poate fi folosita sa schimbi valorile `rgb` pentru culorile definite de `curses` initial. Sa zicem ca vrei sa scazi putin intensitatea culorii rosu. Atunci poti folosi aceasta functie astfel

```

init_color(COLOR_RED, 700, 0, 0);
/* param 1      : numele culorii
 * param 2, 3, 4 : continutul rgb min = 0, max = 1000 */

```

Daca terminalul vostru nu poate schimba definitiile de culoare, functia returneaza ERR. Functia `can_change_color()` poate fi folosita pentru a afla daca terminalul are capacitatea de a schimba continutul culorii sau nu. Continutul rgb este pe o scala de la 0 la 1000. Initial culoare RED (rosu) are definit continutul 1000(r), 0(g), 0(b).

## Continutul culorii

Functiile `color_content()` si `pair_content()` pot fi folosite pentru a afla continutul culorii si combinatia prim-plan, fundal a perechii.

## Administrarea tastelor. Cum citim functiile tastelor, ale ale tastelor sageti, etc..

### Lucruri de baza

Nici un GUI nu este complet fara o puternica interfata cu utilizatorul si care sa interactioneze cu utilizatorul, un program in curses ar trebui sa fie sensibil la apasarea tastelor sau actiunile mouse-ului realizate de utilizator. Sa discutam mai intai despre tastele.

Dupa cum ati vazut in aproape toate exemplele de mai sus, este foarte usor sa obti tasta apasata de utilizator. Un mod simplu de a obtine tastele apasate este sa folosesti functia `getch()`. Modul `cbreak()` ar trebui sa fie activat pentru a citi tastele cand esti interesat sa citesti tastele apasate individual decat linii complete de text(care de obicei sfarsesc prin enter). `keypad` ar trebui sa fie activat pentru a lua functiile tastelor, a tastelor sageti etc. Vedeti sectiunea de initializare pentru detalii.

`getch()` returneaza un intreg corespunzator cu tasta apasata. Daca este un caracter normal, valoarea intreaga va fi echivalenta cu caracterul. Altfel va returna un numar care se poate potrivi cu constantele definite in `curses.h`. Daca de exemplu utilizatorul apasa F1, intregul returnat este 265. Aceasta poate fi verificat folosind macroul `KEY_F()` definita in `curses.h`. Aceasta face citirea tastelor portabila si usor de utilizat.

De exemplu, daca apelati `getch()` astfel

```
int ch;

ch = getch();
```

`getch()` asteapta utilizatorul sa apese o tasta, ( asta daca nu ati specificat un timp limita) si cand utilizatorul apasa o tasta, intregul corespunzator este returnat. Apoi poti verifica intregul returnat cu constantele definite in `curses.h` pentru a vedea daca se potrivesc.

Urmatoarea bucata de cod face acest lucru.

```
if(ch == KEY_LEFT)
    printw("Left arrow is pressed\n");
```

Sa scriem un mic program care creaza un meniu care poate fi navigat cu sagetile sus si jos.

## Codul

```
/* File path: basics/simple_key.c */
#include <stdio.h>
#include <ncurses.h>

#define WIDTH 30
#define HEIGHT 10

int startx = 0;
int starty = 0;

char *choices[] = {
    "Choice 1",
    "Choice 2",
    "Choice 3",
    "Choice 4",
    "Exit",
};

int n_choices = sizeof(choices) / sizeof(char *);
void print_menu(WINDOW *menu_win, int highlight);

int main()
{
    WINDOW *menu_win;
    int highlight = 1;
    int choice = 0;
    int c;

    initscr();
    clear();
    noecho();
    cbreak(); /* Pastrarea liniei este dezactivata. Paseaza totul mie. */
    startx = (80 - WIDTH) / 2;
    starty = (24 - HEIGHT) / 2;

    menu_win = newwin(HEIGHT, WIDTH, starty, startx);
    keypad(menu_win, TRUE);
    mvprintw(0, 0, "Use arrow keys to go up and down, Press enter to
select a choice");
    refresh();
    print_menu(menu_win, highlight);
    while(1)
    {
        c = wgetch(menu_win);
        switch(c)
        {
            case KEY_UP:
                if(highlight == 1)
                    highlight = n_choices;
                else
                    --highlight;
                break;
        }
    }
}
```

```

        case KEY_DOWN:
            if(highlight == n_choices)
                highlight = 1;
            else
                ++highlight;
            break;
        case 10:
            choice = highlight;
            break;
        default:
            mvprintw(24, 0, "Charcter pressed is = %3d
Hopefully it can be printed as '%c'", c, c);
            refresh();
            break;
    }
    print_menu(menu_win, highlight);
    if(choice != 0)/* Utilizatorul a facut alegerea de a iesi
din bucla infinita*/
        break;
    }
    mvprintw(23, 0, "You chose choice %d with choice string %s\n",
choice, choices[choice - 1]);
    clrtoeol();
    refresh();
    endwin();
    return 0;
}

void print_menu(WINDOW *menu_win, int highlight)
{
    int x, y, i;

    x = 2;
    y = 2;
    box(menu_win, 0, 0);
    for(i = 0; i < n_choices; ++i)
    {
        if(highlight == i + 1) /* Alegerea curenta de iluminare*/
        {
            wattron(menu_win, A_REVERSE);
            mvwprintw(menu_win, y, x, "%s", choices[i]);
            wattroff(menu_win, A_REVERSE);
        }
        else
            mvwprintw(menu_win, y, x, "%s", choices[i]);
        ++y;
    }
    wrefresh(menu_win);
}

```

# Interactiunea cu mouse-ul

Acum ca ai vazut cum poti obtine tastele, sa facem acelasi lucru pentru mouse. De obicei fiecare UI permite utilizatorului sa interactionare atat cu mouse-ul cat si cu tastatura.

## Lucruri de baza

Inainte sa faceti altceva, evenimentele pe care vreti sa le primiti trebuie sa fie activate cu `mousemask()`.

```
mousemask(mmask_t newmask, /* Evenimentele pe care vrei sa le
asculti */
          mmask_t *oldmask) /* Vechea masca de evenimente */
```

Primul parametru al functiei de mai sus este o masca de biti a evenimentului pe care vrei sa il asculti. Implicit, toate evenimentele sunt dezactivate. Masca de biti `ALL_MOUSE_EVENTS` poate fi folosita pentru a obtine toate evenimentele.

Urmatoarele sunt toate mastile de evenimente:

| Nume                   | Descriere                                                 |
|------------------------|-----------------------------------------------------------|
| BUTTON1_PRESSED        | mouse buton 1 jos                                         |
| BUTTON1_RELEASED       | mouse buton 1 sus                                         |
| BUTTON1_CLICKED        | mouse buton 1 apasat                                      |
| BUTTON1_DOUBLE_CLICKED | mouse buton 1 apasat dublu                                |
| BUTTON1_TRIPLE_CLICKED | mouse buton 1 apasat triplu                               |
| BUTTON2_PRESSED        | mouse buton 2 jos                                         |
| BUTTON2_RELEASED       | mouse buton 2 sus                                         |
| BUTTON2_CLICKED        | mouse buton 2 apasat                                      |
| BUTTON2_DOUBLE_CLICKED | mouse buton 2 apasat dublu                                |
| BUTTON2_TRIPLE_CLICKED | mouse buton 2 apasat triplu                               |
| BUTTON3_PRESSED        | mouse buton 3 jos                                         |
| BUTTON3_RELEASED       | mouse buton 3 sus                                         |
| BUTTON3_CLICKED        | mouse buton 3 apasat                                      |
| BUTTON3_DOUBLE_CLICKED | mouse buton 3 dublu apasat                                |
| BUTTON3_TRIPLE_CLICKED | mouse buton 3 triplu apasat                               |
| BUTTON4_PRESSED        | mouse buton 4 jos                                         |
| BUTTON4_RELEASED       | mouse buton 4 sus                                         |
| BUTTON4_CLICKED        | mouse buton 4 apasat                                      |
| BUTTON4_DOUBLE_CLICKED | mouse buton 4 dublu apasat                                |
| BUTTON4_TRIPLE_CLICKED | mouse buton 4 triplu apasat                               |
| BUTTON_SHIFT           | shift a fost jos in timp ce starea butonului s-a schimbat |
| BUTTON_CTRL            | CTRL a fost jos in timp ce starea butonului s-a schimbat  |
| BUTTON_ALT             | alt a fost in timp ce starea butonului s-a schimbat       |
| ALL_MOUSE_EVENTS       | raporteaza toate schimbarile de stare a butoanelor        |
| REPORT_MOUSE_POSITION  | raporteaza miscarea mouse-ului                            |

## Obtinerea evenimentelor

O data ce o clasa de evenimente pentru mouse a fost activata, clasa de functii getch() returneaza KEY\_MOUSE de fiecare data cand are loc un eveniment de mouse. Apoi evenimentul de mouse poate fi obtinut cu getmouse().

Codul arata aproximativ astfel:

```
MEVENT event;

ch = getch();
if(ch == KEY_MOUSE)
    if(getmouse(&event) == OK)
        .        /* Fa ceva cu evenimentul */
        :
        .
```

getmouse() returneaza evenimentul in pointerul dat ei. Este o structura care contine

```
typedef struct
{
    short id;          /* ID pentru a distinge mai multe
dispozitive*/
    int x, y, z;      /* coordonatele evenimentelor */
    mmask_t bstate;  /* bitii starii butoanelor */
}
```

bstate este variabila care ne intereseaza. Ea spune starea butonului mouse-ului.

Apoi cu o bucata de cod ca urmatorul, putem afla ce s-a intamplat.

```
if(event.bstate & BUTTON1_PRESSED)
    printf("Left Button Pressed");
```

Aceasta cam este interactiunea cu mouse-ul. Sa cream acelasi meniu si sa activam interactiunea cu mouse-ul. Pentru a face lucrurile mai simple, folosirea tastelor este eliminat.

## Exemplu de interactiune cu mouse-ul

```
/* File Path: basics/mouse_menu.c */
#include <ncurses.h>

#define WIDTH 30
#define HEIGHT 10

int startx = 0;
int starty = 0;

char *choices[] = {
    "Choice 1",
    "Choice 2",
    "Choice 3",
```

```

        "Choice 4",
        "Exit",
    };

int n_choices = sizeof(choices) / sizeof(char *);

void print_menu(WINDOW *menu_win, int highlight);
void report_choice(int mouse_x, int mouse_y, int *p_choice);

int main()
{
    int c, choice = 0;
    WINDOW *menu_win;
    MEVENT event;

    /* Initializeaza curses */
    initscr();
    clear();
    noecho();
    cbreak();          //Pastrarea liniei in buffer dezactivata. Paseaza
//totul mie
    /* Incearca sa puna fereastra in centrul ecranului */
    startx = (80 - WIDTH) / 2;
    starty = (24 - HEIGHT) / 2;

    attron(A_REVERSE);
    mvprintw(23, 1, "Click on Exit to quit");
    refresh();
    attroff(A_REVERSE);

    /* Print the menu for the first time */
    menu_win = newwin(HEIGHT, WIDTH, starty, startx);
    print_menu(menu_win, 1);
    /* Get all the mouse events */
    mousemask(ALL_MOUSE_EVENTS, NULL);

    while(1)
    {
        c = wgetch(menu_win);
        switch(c)
        {
            case KEY_MOUSE:
                if(getmouse(&event) == OK)
                {
                    /* Cand utilizatorul apasa pe buton stang al
mouse-ului */
                    if(event.bstate & BUTTON1_PRESSED)
                    {
                        report_choice(event.x + 1, event.y +
1, &choice);

                        if(choice == -1) //iesire mergi la
sfarsit
                            mvprintw(22, 1, "Choice made is : %d
String Chosen is \"%10s\"", choice, choices[choice - 1]);
                            refresh();
                    }
                }
                print_menu(menu_win, choice);
                break;
        }
    }
}

```

```

end:
    endwin();
    return 0;
}

void print_menu(WINDOW *menu_win, int highlight)
{
    int x, y, i;

    x = 2;
    y = 2;
    box(menu_win, 0, 0);
    for(i = 0; i < n_choices; ++i)
    {
        if(highlight == i + 1)
        {
            wattron(menu_win, A_REVERSE);
            mvwprintw(menu_win, y, x, "%s", choices[i]);
            wattroff(menu_win, A_REVERSE);
        }
        else
            mvwprintw(menu_win, y, x, "%s", choices[i]);
        ++y;
    }
    wrefresh(menu_win);
}

/* Raporteaza alegerea in functie de pozitia mouse-ului */
void report_choice(int mouse_x, int mouse_y, int *p_choice)
{
    int i, j, choice;

    i = startx + 2;
    j = starty + 3;

    for(choice = 0; choice < n_choices; ++choice)
        if(mouse_y == j + choice && mouse_x >= i && mouse_x <= i +
strlen(choices[choice]))
        {
            if(choice == n_choices - 1)
                *p_choice = -1;
            else
                *p_choice = choice + 1;
            break;
        }
}

```

## Variate functii

Functiile `mouse_trafo()` si `wmouse_trafo()` pot fi folosite pentru a converti de la coordonatele mouse-ului la coordonatele relative ale ferestrei. Vezi in pagina principala  `curs_mouse(3X)` pentru detalii.

Functia `mouseinterval` seteaza timpul maxim( in mii de secunde) care poate trece intre apasarea si eliberarea evenimentelor pentru ca ele sa fie recunoscute ca click. Aceasta functi returneaza valoarea anterioara a intervalului. Implicit este o cincime dintr-o secunda.

## Administrarea ecranului

In aceasta sectiune, vom vedea cateva functii, care ne permit sa folosim eficient ecranul si sa scriem program interesante. Aceasta este in mod special important in scrierea jocurilor.

### Functiile `getyx()`

Functia `getyx()` poate fi folosita pentru a afla coordonatele curente ale cursorului. Va pune valorile coordonatelor `x` si `y` in argumentele date ei. Din moment ce `getyx()` este un macro nu trebuie sa transmiteti adresele variabilelor. Poate fi apelata astfel:

```
getyx(win, y, x);
/* win: pointer fereastră
 *   y, x: coordonatele y, x vor fi puse in aceste variabile */
```

Functia `getparyx()` obtine inceputul coordonatelor sub-ferestrei relativ la fereastră principala. Aceasta este uneori utila pentru a actualiza sub-ferestra. Cand proiectam chestii interesante cum ar fi meniurile multiple, devine dificil de memorat pozitiile meniurilor, coordonatele primei optiuni etc. O solutie simpla la aceasta problema, este sa cream meniuri in sub-ferestre si mai tarziu sa aflam coordonatele de inceput ale meniului utilizand `getparyx()`.

Functiile `getbegyx()` si `getmaxyx()` memoreaza coordonatele de inceput si cele maxime ale ferestrei. Aceste functii sunt utile in acelai fel ca mai sus in utilizarea efectiva a ferestrelor si asub-ferestrelor.

## Descarcarea ecranului

In timp ce scriem jocuri, cateodata devine necesar sa memoram starea ecranului si sa o restauram in aceeasi stare. Functia `scr_dump()` poate fi folosita pentru a descarca continutul ecranului intr-un fisier dat ca argument. Mai tarziu poate fi restaurat prin functia `scr_restore`. Aceste 2 functii simple pot fi folosite pentru a mentine o miscare rapida a jocului cu schimbari de scenarii.

## Descarcarea ferestrei

Pentru a descarca si a restaura ferestre, functiile `putwin()` si `getwin()` pot fi folosite. `Putwin()` pune starea curenta a ferestrei intr-un fisier, care poate fi mai tarziu restaurat cu `getwin()`.

Functia `copywin()` poate fi folosita pentru a copia o fereastră complet in alta fereastră. Ea ia ferestrele sursa si destinatie ca parametri in acord cu dreptunghiul specificat, copie regiunea dreptunghiului de la fereastră sursa la destinatie. Ultimul ei parametru specifica daca sa se suprascrise sau doar sa se acopere continutul pe fereastră destinatie. Daca argumentul este true, atunci copierea este nedestructiva.

## Functii variate

Acum stiti destule functii pentru a scrie un program bun in curses, cu toate sunetele si llicaririle. Exista cateva functii variate care sunt utile in diferite cazuri. Sa vedem cateva dintre acestea.

`curs_set()`

Aceasta functie poate fi folosita ca sa facem cursorul invizibil. Parametrul acestei funtii trebuie sa fie:

```
0 : invisibil or
1 : normal or
2 : foarte vizibil.
```

## Parasirea temporara a modului curses

Cateodata ai dori sa revii la modul cooked(modul normal fara retinerea liniei intr-un buffer) temporar. In acest caz trebuie sa salvezi modurile tts cu un apel `def_prog_mode()` si apoi apelul `endwin()` pentru a termina modul curses. Aceasta readuce modul tty. Pentru a reveni la modul curses o data ce ai terminat, apeleaza `reset_prog_mode()`. Aceasta functie returneaza tty starii stocate de `def_prog_mode()`. Apoi apeleaza `refresh()`, si esti inapoi in modul curses. Aici este un exemplu care arata secventa de lucruri care trebuie urmata.

```
/* File Path: basics/temp_leave.c */
#include <ncurses.h>

int main()
{
    initscr(); /* Incepe modul curses */
   printw("Hello World !!!\n"); /* Afiseaza Hello World */
    refresh(); /* Il afiseaza pe ecranul real */
    def_prog_mode(); /* Salveaza modurile tty */
    endwin(); /* Termina modul curses temporar */
    system("/bin/sh"); /* Fa ce doresti in modul cooked */
    reset_prog_mode(); /* Revina ma modul tty anterior*/
    /* stocat de def_prog_mode() */
    refresh(); /* Fa refresh() pentru a restaura */
    /* continutul ecranului */
    printw("Another String\n"); /* Inapoi la curses foloseste in
totalitate */
    refresh(); /* capacitatile curses */
    endwin(); /* sfarsit mod curses */

    return 0;
}
```

# Alte Librarii

In afara de libraria curses, exista cateva librarii mod text, care furnizeaza mai multa functionalitate si o multime de caracteristici. Acesta sectiune explica trei librarii standard care sunt de obicei distribuite impreuna cu curses.

## Libraria Panel

Acum ca sunteti buni in curses, vreti sa faceti ceva mare. Ati creat o multime de ferestre acoperite una pe alta pentru a da o imagine profesionala de tipuri de ferestre. Din pacate, curand devine dificil sa lucrezi cu ele. Multiplele actualizari te duc spre un cosmar. Ferestrele suprapuse creaza urme, oricand uiti sa actualizezi ferestrele in ordinea corecta.

Nu dispera. Exista o solutie eleganta oferita de libraria panel. Asa cum spun proiectantii de ncurses

*Cand proiectarea interfetei tale este astfel incat ferestrele pot cufunda mai adanc in stiva de vizibilitate sau pot iesi la suprafata la executie, rezultatul ca la carte poate dura mult si poate fi dificil de obtinut. De acum inainte librariile panel.*

Daca aveti multe ferestre suprapuse, atunci libraria panel este solutia. Aceasta eludeaza nevoia de a face o serie de wnoutrefresh(), douupdate() si arata greutatea de a le face corect(jos sus). Libraria mentine informatii despre ordinea ferestrelor, suprapunerea lor si actualizeaza ecranul corect. Asadar ce mai asteptam? Sa aruncam o privire de aproape in paneluri.

## Lucruri de baza

Obiectul Panel este o fereastră este tratat implicit ca o parte a unei punti incluzand toate celelalte obiecte panel. Puntea este tratata ca o stiva cu panelul de sus complet vizibil pot fi sau nu obscure in functie de pozitia lor. Deci idea de baza este sa creezi o stiva de paneluri suprapuse si sa folosesti libraria de paneluri pentru a le afisa corect. Exista o functie similara cu refresh() care, cand este apelata, afiseaza panelurile in ordinea corecta. Functiile pot ascunde sau afisa panelurile, muta panelurile, schimba dimensiunea lor etc.. Suprapunerea este o problema administrata de libraria panel in timpul tuturor apelurilor acestor functii.

Cursul general al unui program panel merge astfel:

1. Creaza fereastră care sa fie atasata panelului(cu newwin())
2. Creaza un panel cu ordinea de vizibilitate aleasa. Pune-le in stiva in acord cu vizibilitatea dorita. Functia new\_panel() este folosita pentru a crea paneluri.
3. Apeleaza update\_panels() pentru a scrie panelurile pe ecranul virtual in ordinea de vizibilitate corecta. Apeleaza douupdate() pentru a afisa pe ecran.
4. Folositi panelurile cu show\_panel(), hide\_panel(), move\_panel() etc. Folositi-va de functii ca panel\_hidden() si panel\_window(). Folositi pointeri utilizatori pentru a

memora date specifice panelurilor. Folositi functiile `set_panel_userptr()` and `panel_userptr()` pentru a seta si a obtine pointerul utilizator a panelului.

5. Cand ai terminat cu panelul foloseste `del_panel()` pentru a sterge panelul.

Sa clarificam conceptele, cu cateva programe. Urmatorul este un program simplu care creaza 3 paneluri suprapuse si le afiseaza pe ecran.

## Compilarea cu Libraria Panels

Pentru a folosi functiile libreriei `panel`, trebuie sa incluzi `panel.h` si pentru a lega programul cu libraria `panel` ar trebui adaugat steguletul `-lpanel` impreuna cu `-lncurses` in aceasta ordine.

```
#include <panel.h>
.
.
.

compilare si legare: gcc <program file> -lpanel -lncurses
```

## Un exemplu simplu

```
/* File Path: panels/panel_simple.c */
#include <panel.h>

int main()
{
    WINDOW *my_wins[3];
    PANEL *my_panels[3];
    int lines = 10, cols = 40, y = 2, x = 4, i;

    initscr();
    cbreak();
    noecho();

    /* Creaza o noua fereastră pentru panel */
    my_wins[0] = newwin(lines, cols, y, x);
    my_wins[1] = newwin(lines, cols, y + 1, x + 5);
    my_wins[2] = newwin(lines, cols, y + 2, x + 10);

    /*
     * Creaza margini in jurul ferestrelor pentru ca sa se vad efectul
     panelulilor. */
    for(i = 0; i < 3; ++i)
        box(my_wins[i], 0, 0);

    /* Ataseaza un panel la fiecare fereastră */
    /* Ordinea este sus jos */
    my_panels[0] = new_panel(my_wins[0]); /* Pune 0, ordine: stdscr-0 */
    my_panels[1] = new_panel(my_wins[1]); /* Pune 1,ordine: stdscr-0-1*/
    my_panels[2] = new_panel(my_wins[2]); /* Pune 2,ordine:stdscr-0-1-2
*/

    /* Actualizeaza ordinea de stiva. Al doilea panel va fi deasupra */
    update_panels();
```

```

    /* Le afiseaza pe ecran */
    doupdate();

    getch();
    endwin();
}

```

Dupa cum vedeti programul de mai sus urmeaza un curs simplu fiind explicat. Ferestrele sunt create cu `newwin()` si sunt atasate panelurilor cu `new_panel()`. Pe masura ce atasam paneluri unul dupa altul, stiva de paneluri se actualizeaza. Pentru a le afisa sunt apelate `update_panels` si `doupdate()`.

Un exemplu putin mai complicat este dat mai jos. Programul creaza 3 ferestre care pot fi ciclitate folosind `tab`. Aruncati o privire asupra codului.

## Un exemplu Parcurgerea unei ferestre

```

/* File Path: panels/panel_browse.c */
#include <panel.h>

#define NLINES 10
#define NCOLS 40

void init_wins(WINDOW **wins, int n);
void win_show(WINDOW *win, char *label, int label_color);
void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color);

int main()
{
    WINDOW *my_wins[3];
    PANEL *my_panels[3];
    PANEL *top;
    int ch;

    /* Initialize curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* Initialize all the colors */
    init_pair(1, COLOR_RED, COLOR_BLACK);
    init_pair(2, COLOR_GREEN, COLOR_BLACK);
    init_pair(3, COLOR_BLUE, COLOR_BLACK);
    init_pair(4, COLOR_CYAN, COLOR_BLACK);

    init_wins(my_wins, 3);

    /* Attach a panel to each window */ /* Ordinea este jos sus */
    my_panels[0] = new_panel(my_wins[0]); /* Pune 0, ordine: stdscr-0 */
    my_panels[1] = new_panel(my_wins[1]); /* Pune 1, ordine: stdscr-0-1

```

```

*/
my_panels[2] = new_panel(my_wins[2]); /* Pune 2, ordine: stdscr-0-1-
2 */

/* Seteaza pointerii utilizator catre urmatorul panel */
set_panel_userptr(my_panels[0], my_panels[1]);
set_panel_userptr(my_panels[1], my_panels[2]);
set_panel_userptr(my_panels[2], my_panels[0]);

/* Updateaza ordinea stivei. Al doilea panel va fi deasupra */
update_panels();

/* Il afiseaza pe ecran */
attron(COLOR_PAIR(4));
mvprintw(LINES - 2, 0, "Use tab to browse through the windows (F1 to
Exit)");
attroff(COLOR_PAIR(4));
doupdate();

top = my_panels[2]; /* Memoreaza panelul de deasupra */
while((ch = getch()) != KEY_F(1))
{
    switch(ch)
    {
        case 9:
            top = (PANEL *)panel_userptr(top); /* Gaseste
urmatorul panel din ciclu */
            top_panel(top); /* Il face ca fiind panelul
de deasupra */
            break;
    }
    update_panels();
    doupdate();
}
endwin();
return 0;
}

/*Pune toate ferestrele */
void init_wins(WINDOW **wins, int n)
{
    int x, y, i;
    char label[80];

    y = 2;
    x = 10;
    for(i = 0; i < n; ++i)
    {
        wins[i] = newwin(NLINES, NCOLS, y, x);
        sprintf(label, "Window Number %d", i + 1);
        win_show(wins[i], label, i + 1);
        y += 3;
        x += 7;
    }
}

/* Afiseaza fereastra cu o margine si un label */
void win_show(WINDOW *win, char *label, int label_color)
{
    int startx, starty, height, width;

```

```

    getbegyx(win, starty, startx);
    getmaxyx(win, height, width);

    box(win, 0, 0);
    mvwaddch(win, 2, 0, ACS_LTEE);
    mvwhline(win, 2, 1, ACS_HLINE, width - 2);
    mvwaddch(win, 2, width - 1, ACS_RTEE);

    print_in_middle(win, 1, 0, width, label, COLOR_PAIR(label_color));
}

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color)
{
    int length, x, y;
    float temp;

    if(win == NULL)
        win = stdscr;
    getyx(win, y, x);
    if(startx != 0)
        x = startx;
    if(starty != 0)
        y = starty;
    if(width == 0)
        width = 80;

    length = strlen(string);
    temp = (width - length) / 2;
    x = startx + (int)temp;
    wattron(win, color);
    mvwprintw(win, y, x, "%s", string);
    wattroff(win, color);
    refresh();
}

```

## Folosirea pointerilor utilizator

In exemplul de mai sus am folosit pointeri utilizatori pentru a afla urmatorul panel din ciclu. Putem atasa informatii specifice panelurilor prin specificarea unui pointer utilizator, care arata orice informatie pe care vrei sa o stochezi. In acest caz am pastrat pointerul spre urmatorul panel din ciclu. Pointerul utilizator pentru un panel poate fi setat cu functia `set_panel_userptr()`. Poate fi accesat folosind functia `panel_userptr()` care va returna pointerul utilizator pentru panelul dat ca argument. Dupa aflarea urmatorului panel din ciclu acesta este pus deasupra de functia `top_panel()`. Aceasta functie pune panelul dat ca argument in varful stivei de paneluri.

## Mutarea si redimensionarea panelurilor

Functia `move_panel` poate fi folosita pentru a muta un panel la locatia dorita. Ea nu schimba pozitia panelului in stiva. Ai grija sa folosesti `move_panel()` in loc de `mvwin()` in fereastra asociata panelului.

Redimensionarea unui panel este destul de complicata. Nu exista o functie care sa redimensioneze direct fereastra asociata panelului. O solutie pentru a redimensiona fereastra este sa cream o noua fereastra cu dimensiunea dorita, schimba fereastra asociata cu panelul folosind `replace_panel()`. Nu uita sa stergi vechea fereastra. Fereastra asociata cu panelul poate fi gasita folosind functia `panel_window()`.

Urmatorul program arata aceste concepte, intr-un program presupus a fi simplu. Poti cicla in fereastra cu TAB ca de obicei. Pentru a redimensiona sau muta panelul activ apasa 'r' pentru redimensionare si 'm' pentru mutare. Acest exemplu foloseste date de la utilizator pentru a obtine datele cerute pentru a face operatiile.

## Exemplu de mutare si redimensionare

```
/* File Path: panels/panel_resize.c */
#include <panel.h>;

typedef struct _PANEL_DATA {
    int x, /* Startx */
    y, /* Starty */
    w, /* Width */
    h; /* Height */
    char label[80]; /* Label pentru fereastra */
    int label_color; /* Numarul culorii petru label */
    PANEL *next; /* Pointer spre urmatorul panel din ciclu */
}PANEL_DATA;

#define NLINES 10
#define NCOLS 40

void init_wins(WINDOW **wins, int n);
void win_show(WINDOW *win, char *label, int label_color);
void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color);
void set_user_ptrs(PANEL **panels, int n);

int main()
{
    WINDOW *my_wins[3];
    PANEL *my_panels[3];
    PANEL_DATA *top;
    PANEL *stack_top;
    WINDOW *temp_win, *old_win;
    int ch;
    int newx, newy, neww, newh;
    int size = FALSE, move = FALSE;

    /* Initialize curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* Initializeaza toate culorile */
```

```

init_pair(1, COLOR_RED, COLOR_BLACK);
init_pair(2, COLOR_GREEN, COLOR_BLACK);
init_pair(3, COLOR_BLUE, COLOR_BLACK);
init_pair(4, COLOR_CYAN, COLOR_BLACK);

init_wins(my_wins, 3);

/* Attach a panel to each window */ /* Ordinea este jos sus */
my_panels[0] = new_panel(my_wins[0]); /* Pune 0,ordine: stdscr-0 */
my_panels[1] = new_panel(my_wins[1]); /*Pune 1,ordine: stdscr-0-1 */
my_panels[2] = new_panel(my_wins[2]); /* Pune 2,ordine: stdscr-0-1-2
*/

set_user_ptrs(my_panels, 3);
/* Updateaza ordinea stivei.Al doilea panel va fi deasupra */
update_panels();

/* Il afiseaza pe ecran */
attron(COLOR_PAIR(4));
mvprintw(LINES - 2, 0, "Use tab to browse through the windows (F1 to
Exit)");
attroff(COLOR_PAIR(4));
doupdate();

stack_top = my_panels[2];
top = (PANEL_DATA *)panel_userptr(stack_top);
newx = top->x;
newy = top->y;
neww = top->w;
newh = top->h;
while((ch = getch()) != KEY_F(1))
{
    switch(ch)
    {
        case 9: /* Tab */
            top = (PANEL_DATA *)panel_userptr(stack_top);
            top_panel(top->next);
            stack_top = top->next;
            top = (PANEL_DATA *)panel_userptr(stack_top);
            newx = top->x;
            newy = top->y;
            neww = top->w;
            newh = top->h;
            break;

        case 'r': /* Redimensionare*/
            size = TRUE;
            attron(COLOR_PAIR(4));
            mvprintw(LINES - 3, 0, "Entered Resizing :Use
Arrow Keys to resize and press <ENTER> to end resizing");
            refresh();
           attroff(COLOR_PAIR(4));
            break;

        case 'm': /* Mutare */
            attron(COLOR_PAIR(4));
            mvprintw(LINES - 3, 0, "Entered Moving: Use
Arrow Keys to Move and press <ENTER> to end moving");
            refresh();
           attroff(COLOR_PAIR(4));

```

```

        move = TRUE;
        break;
    case KEY_LEFT:
        if(size == TRUE)
        {
            --newx;
            ++neww;
        }
        if(move == TRUE)
            --newx;
        break;
    case KEY_RIGHT:
        if(size == TRUE)
        {
            ++newx;
            --neww;
        }
        if(move == TRUE)
            ++newx;
        break;
    case KEY_UP:
        if(size == TRUE)
        {
            --newy;
            ++newh;
        }
        if(move == TRUE)
            --newy;
        break;
    case KEY_DOWN:
        if(size == TRUE)
        {
            ++newy;
            --newh;
        }
        if(move == TRUE)
            ++newy;
        break;
    case 10: /* Enter */
        move(LINES - 3, 0);
        clrtoeol();
        refresh();
        if(size == TRUE)
        {
            old_win = panel_window(stack_top);
            temp_win = newwin(newh, neww, newy,
newx);

            replace_panel(stack_top, temp_win);
            win_show(temp_win, top->label, top-
>label_color);

            delwin(old_win);
            size = FALSE;
        }
        if(move == TRUE)
        {
            move_panel(stack_top, newy, newx);
            move = FALSE;
        }
        break;
}
update_panels();

```

```

        douupdate();
    }
    endwin();
    return 0;
}

/* Put all the windows */
void init_wins(WINDOW **wins, int n)
{
    int x, y, i;
    char label[80];

    y = 2;
    x = 10;
    for(i = 0; i < n; ++i)
    {
        wins[i] = newwin(NLINES, NCOLS, y, x);
        sprintf(label, "Window Number %d", i + 1);
        win_show(wins[i], label, i + 1);
        y += 3;
        x += 7;
    }
}

/* Seteaza structurile PANEL_DATA pentru fiecare panel in parte */
void set_user_ptrs(PANEL **panels, int n)
{
    PANEL_DATA *ptrs;
    WINDOW *win;
    int x, y, w, h, i;
    char temp[80];

    ptrs = (PANEL_DATA *)calloc(n, sizeof(PANEL_DATA));

    for(i = 0; i < n; ++i)
    {
        win = panel_window(panels[i]);
        getbegyx(win, y, x);
        getmaxyx(win, h, w);
        ptrs[i].x = x;
        ptrs[i].y = y;
        ptrs[i].w = w;
        ptrs[i].h = h;
        sprintf(temp, "Window Number %d", i + 1);
        strcpy(ptrs[i].label, temp);
        ptrs[i].label_color = i + 1;
        if(i + 1 == n)
            ptrs[i].next = panels[0];
        else
            ptrs[i].next = panels[i + 1];
        set_panel_userptr(panels[i], &ptrs[i]);
    }
}

/* Afiseaza fereastra cu margini si label */
void win_show(WINDOW *win, char *label, int label_color)
{
    int startx, starty, height, width;

    getbegyx(win, starty, startx);
    getmaxyx(win, height, width);

```

```

        box(win, 0, 0);
        mvwaddch(win, 2, 0, ACS_LTEE);
        mvwhline(win, 2, 1, ACS_HLINE, width - 2);
        mvwaddch(win, 2, width - 1, ACS_RTEE);

        print_in_middle(win, 1, 0, width, label, COLOR_PAIR(label_color));
    }

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color)
{
    int length, x, y;
    float temp;

    if(win == NULL)
        win = stdscr;
    getyx(win, y, x);
    if(startx != 0)
        x = startx;
    if(starty != 0)
        y = starty;
    if(width == 0)
        width = 80;

    length = strlen(string);
    temp = (width - length) / 2;
    x = startx + (int)temp;
    watttrn(win, color);
    mvwprintw(win, y, x, "%s", string);
    wattroff(win, color);
    refresh();
}

```

Concentrați-vă asupra buclei while principale. O dată aflată tasta apăsată, ia acțiunea corespunzătoare. Dacă este apăsată 'r' este început modul resizing. Apoi noile dimensiuni sunt actualizate o dată ce utilizatorul apasă tastele săgeți. Când utilizatorul apasă <Enter> se termină selecția curentă și panelul este redimensionat utilizând conceptul explicat. Este lăsat ca exercițiu pentru cititor de a afișa border punctat pe măsură ce se redimensionează la o nouă poziție.

Când utilizatorul apasă 'm' începe modul move. Aceasta este mai simplă puțin decât redimensionarea. Pe măsură ce sunt apăsate tastele săgeți noua poziție este actualizată și apăsarea <Enter> face ca panelul să fie mutat prin apelul funcției move\_panel().

În acest program datele utilizator care sunt reprezentate ca PANEL\_DATA, joacă un rol foarte important în găsirea informațiilor asociate cu panelul. Așa cum scrie și în comentarii, PANEL\_DATA memorează dimensiunea panelului, labelul, culoarea labelului și un pointer la următorul panel din ciclu.

## Ascunderea si afisarea panelurilor

Un panel poate fi ascuns folosind functia `hide_panel()`. Aceasta functie doar il scoate din stiva de paneluri, astfel ascunzandu-l de pe ecran se fac atunci cand apelam `update_panels()` si `doupdate()`. Aceasta nu distruge structura `PANEL` asociata panelului ascuns. Se poate reafisa folosind functia `show_panel()`.

Urmatorul program arata ascunderea panelurilor. Apasati 'a' sau 'b' sau 'c' pentru a afisa sau a ascunde prima, a doua, respectiv a treia fereastră. Foloseste o data utilizator cu o variabila `hide`, care retine daca fereastra este ascunsa sau nu. Dintr-un anumit motiv functia `panel_hidden()` care spune daca fereastra este ascunsa sau nu nu functioneaza. Un raport cu greseala a fost de asemenea prezentat de Michael Andres [aici.](#)

## Ascunderea si afisarea panelurilor Exemplu

```
/* File Path: panels/panel_hide.c */
#include <panel.h>

typedef struct _PANEL_DATA {
    int hide;          /* TRUE daca panelul este ascuns */
}PANEL_DATA;

#define NLINES 10
#define NCOLS 40

void init_wins(WINDOW **wins, int n);
void win_show(WINDOW *win, char *label, int label_color);
void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color);

int main()
{
    WINDOW *my_wins[3];
    PANEL *my_panels[3];
    PANEL_DATA panel_datas[3];
    PANEL_DATA *temp;
    int ch;

    /* Initializeaza curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* Initializeaza toate culorile */
    init_pair(1, COLOR_RED, COLOR_BLACK);
    init_pair(2, COLOR_GREEN, COLOR_BLACK);
    init_pair(3, COLOR_BLUE, COLOR_BLACK);
    init_pair(4, COLOR_CYAN, COLOR_BLACK);

    init_wins(my_wins, 3);

    /* Ataseaza un panel fiecarei ferestre */ /* Ordinea este sus-jos */
    my_panels[0] = new_panel(my_wins[0]); /* Pune 0, ordinea:stdscr-0 */
```

```

my_panels[1] = new_panel(my_wins[1]); /*Pune 1,ordinea:stdscr-0-1 */
my_panels[2] = new_panel(my_wins[2]); /*Pune 2,ordinea:stdscr-0-1-2
*/

/* Initializeaza datele panel spunand ca nimic nu este ascuns */
panel_datas[0].hide = FALSE;
panel_datas[1].hide = FALSE;
panel_datas[2].hide = FALSE;

set_panel_userptr(my_panels[0], &panel_datas[0]);
set_panel_userptr(my_panels[1], &panel_datas[1]);
set_panel_userptr(my_panels[2], &panel_datas[2]);

/* Actualizeaza ordinea stivei. Al doilea panel va fi in varf */
update_panels();

/* Afiseaza pe ecran */
attron(COLOR_PAIR(4));
mvprintw(LINES - 3, 0, "Show or Hide a window with 'a'(first window)
'b'(Second Window) 'c'(Third Window)");
mvprintw(LINES - 2, 0, "F1 to Exit");

attroff(COLOR_PAIR(4));
doupdate();

while((ch = getch()) != KEY_F(1))
{
    switch(ch)
    {
        case 'a':
            temp = (PANEL_DATA
*)panel_userptr(my_panels[0]);
            if(temp->hide == FALSE)
            {
                hide_panel(my_panels[0]);
                temp->hide = TRUE;
            }
            else
            {
                show_panel(my_panels[0]);
                temp->hide = FALSE;
            }
            break;
        case 'b':
            temp = (PANEL_DATA
*)panel_userptr(my_panels[1]);
            if(temp->hide == FALSE)
            {
                hide_panel(my_panels[1]);
                temp->hide = TRUE;
            }
            else
            {
                show_panel(my_panels[1]);
                temp->hide = FALSE;
            }
            break;
        case 'c':
            temp = (PANEL_DATA
*)panel_userptr(my_panels[2]);
            if(temp->hide == FALSE)
            {
                hide_panel(my_panels[2]);

```

```

        temp->hide = TRUE;
    }
    else
    {
        show_panel(my_panels[2]);
        temp->hide = FALSE;
    }
    break;
}
update_panels();
doupdate();
}
endwin();
return 0;
}

/* Pune toate ferestrele */
void init_wins(WINDOW **wins, int n)
{
    int x, y, i;
    char label[80];

    y = 2;
    x = 10;
    for(i = 0; i < n; ++i)
    {
        wins[i] = newwin(NLINES, NCOLS, y, x);
        sprintf(label, "Window Number %d", i + 1);
        win_show(wins[i], label, i + 1);
        y += 3;
        x += 7;
    }
}

/* Afiseaza fereastra cu un border si un label */
void win_show(WINDOW *win, char *label, int label_color)
{
    int startx, starty, height, width;

    getbegyx(win, starty, startx);
    getmaxyx(win, height, width);

    box(win, 0, 0);
    mvwaddch(win, 2, 0, ACS_LTEE);
    mvwhline(win, 2, 1, ACS_HLINE, width - 2);
    mvwaddch(win, 2, width - 1, ACS_RTEE);

    print_in_middle(win, 1, 0, width, label, COLOR_PAIR(label_color));
}

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color)
{
    int length, x, y;
    float temp;

    if(win == NULL)
        win = stdscr;
    getyx(win, y, x);
    if(startx != 0)
        x = startx;

```

```

    if(starty != 0)
        y = starty;
    if(width == 0)
        width = 80;

    length = strlen(string);
    temp = (width - length)/ 2;
    x = startx + (int)temp;
    wattron(win, color);
    mvwprintw(win, y, x, "%s", string);
    wattroff(win, color);
    refresh();
}

```

### **Funcțiile `panel_above()` și `panel_below()`**

Funcțiile `panel_above` și `panel_below()` pot fi folosite pentru a afla panelul de deasupra și dedesubtul unui panel. Dacă argumentul acestor funcții este NULL, atunci returnează un pointer către panelul de jos și respectiv cel din varf.

## **Biblioteca Menu**

Bibliotecile `menu` furnizează o extensie la lucrurile de bază din `curses`, prin care se pot crea meniuri. Furnizează un set de funcții pentru crearea meniurilor. Dar ele trebuie să fie personalizate pentru a da o imagine mai dragută, cu culori etc. Să intrăm în detalii.

Un meniu este o parte de ecran care folosește utilizatorului pentru a alege un subset dintr-un set dat de itemuri. Pentru simplificare, un meniu este o colecție de itemuri din care unul sau mai mulți pot fi aleși. Unii cititori s-ar putea să nu știe de capacitatea de selecție multiplă a itemurilor. Biblioteca `menu` furnizează funcționalitatea de a scrie meniuri din care utilizatorul poate alege mai mult de un item ca opțiune preferată. Acest lucru este discutat într-o secțiune viitoare. Acum este momentul pentru detalieri.

### **Lucruri de bază**

Pentru a crea meniuri, mai întâi cream itemuri, și apoi postăm meniul pe ecran. După aceea, toate procesările la răspunsurile utilizatorului este realizată într-o funcție elegantă `menu_driver()` care este baza oricărui program cu meniuri.

Cursul general al controlului asupra unui program cu meniuri arată astfel.

1. Inițializează `curses`.
2. Creează itemuri folosind `new_item()`. Poți specifica numele și o descriere pentru itemuri.
3. Creează meniul cu `new_menu()` prin specificarea itemurilor care i se atașează.
4. Postează meniul cu `menu_post()` și actualizează ecranul.
5. Procesează cererile utilizatorului printr-o buclă și realizează actualizările necesare ale meniului cu `menu_driver`.
6. Scoate meniul cu `menu_unpost()`.

7. Elibereaza memoria alocata meniului cu free\_menu().
8. Elibereaza memoria alocata itemurilor cu free\_item().
9. Termina curses.

Sa vedem un program care afiseaza un mniu simplu si care actualizeaza selectia curenta cu sagetile sus, jos.

## Compilarea cu Biblioteca Menu

Pentru a folosi functiile bibliotecii, trebuie sa incluzi menu.h si ca sa legi programul cu biblioteca menu trebuie adaugat steguletul -lmenu impreuna cu -lncurses in aceasta ordine.

```
#include <menu.h>
.
.
.

compilare si legare: gcc <program file> -lmenu -lncurses
```

## Un exemplu simplu

```
/* File Path: menus/menu_simple.c */
#include <menu.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD 4

char *choices[] = {
    "Choice 1",
    "Choice 2",
    "Choice 3",
    "Choice 4",
    "Exit",
};

int main()
{
    ITEM **my_items;
    int c;
    MENU *my_menu;
    int n_choices, i;
    ITEM *cur_item;

    initscr();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    n_choices = ARRAY_SIZE(choices);
    my_items = (ITEM **)calloc(n_choices + 1, sizeof(ITEM *));

    for(i = 0; i < n_choices; ++i)
        my_items[i] = new_item(choices[i], choices[i]);
    my_items[n_choices] = (ITEM *)NULL;
```

```

my_menu = new_menu((ITEM **)my_items);
post_menu(my_menu);
refresh();

while((c = getch()) != KEY_F(1))
{
    switch(c)
    {
        case KEY_DOWN:
            menu_driver(my_menu, REQ_DOWN_ITEM);
            break;
        case KEY_UP:
            menu_driver(my_menu, REQ_UP_ITEM);
            break;
    }
}

free_item(my_items[0]);
free_item(my_items[1]);
free_menu(my_menu);
endwin();
}

```

Acest program demonstreaza conceptele de baza implicate in crearea unui meniu folosind biblioteca meniului. Mai intai cream itemurile cu `new_item()` si apoi le atasam la meniu cu functia `new_menu`. Dupa postarea meniului si actualizarea ecranului, bucla principala de procesare incepe. Ea citeste intrarea de la utilizator si ia actiunea corespunzatoare. Functia `menu_driver()` este baza meniului sistem.

Al doilea parametru al acestei functii este cel ce ne spune ce trebuie facut cu meniul in cauza. In functie de parametru, functia `menu_driver()` indeplineste sarcina corespunzatoare. Valoarea poate fi o cerere de navigare in meniu, un caracter ascii sau o tasta speciala `KEY_MOUSE` asociata unui eveniment al mouse-ului.

Functia `menu_driver` accepta urmatoarele cereri de navigare:

|                             |                                         |
|-----------------------------|-----------------------------------------|
| <code>REQ_LEFT_ITEM</code>  | Deplasare la stanga fata de un articol  |
| <code>REQ_RIGHT_ITEM</code> | Deplasare la dreapta fata de un articol |
| <code>REQ_UP_ITEM</code>    | Deplasare deasupra unui articol         |
| <code>REQ_DOWN_ITEM</code>  | Deplasare dedesuptul unui articol       |
| <code>REQ_SCR_ULINE</code>  | Deplasare cu o linie mai sus            |
| <code>REQ_SCR_DLINE</code>  | Deplasare cu o linie mai jos            |
| <code>REQ_SCR_DPAGE</code>  | Deplasare cu o pagina mai jos           |
| <code>REQ_SCR_UPAGE</code>  | Deplasare cu o pagina mai sus           |

|                   |                                                                    |
|-------------------|--------------------------------------------------------------------|
| REQ_FIRST_ITEM    | Pozitionare pe primul articol                                      |
| REQ_LAST_ITEM     | Pozitionare pe ultimul articol                                     |
| REQ_NEXT_ITEM     | Deplasare la urmatorul articol                                     |
| REQ_PREV_ITEM     | Deplasare la articolul anterior                                    |
| REQ_TOGGLE_ITEM   | Selectare/Deselectare a unui articol                               |
| REQ_CLEAR_PATTERN | Goleste bufferul sablonului meniului                               |
| REQ_BACK_PATTERN  | Sterge caracterul anterior din bufferul sablonului                 |
| REQ_NEXT_MATCH    | Se deplaseaza la urmatorul articol care se potriveste cu sablonul  |
| REQ_PREV_MATCH    | Se deplaseaza la articolul anterior care se potriveste cu sablonul |

Nu fiti coplestiti de numarul de optiuni. Le vom analiza una cate una. Optiunile interesante in acest exemplu sunt REQ\_UP\_ITEM si REQ\_DOWN\_ITEM. Cand menu\_driver primeste aceste optiuni ea actualizeaza articolul curent cu un articol de sus respectiv de jos.

## Menu Driver : Baza meniului sistem

Asa cum ati vazut in exemplul de mai sus, menu\_driver joaca un rol important in actualizarea meniului. Este important de inteles optiunile variate pe care le primeste si ce fac ele.

- *REQ\_LEFT\_ITEM si REQ\_RIGHT\_ITEM*

Un meniu poate fi expus cu mai multe coloane pentru mai multe articole. Acest lucru poate fi realizat folosind functia menu\_format(). Cand un meniu cu mai multe coloane este expus aceste cereri determina functia sa mute selectia curenta la stanga sau la dreapta.

- *REQ\_UP\_ITEM si REQ\_DOWN\_ITEM*

Ati vazut aceste doua optiuni in exemplul de mai sus. Cand sunt transmise ele determina menu\_driver sa mute selectia curenta cu un articol mai jos sau mai sus.

- *REQ\_SCR\_\* options*

Cele patru optiuni REQ\_SCR\_ULINE, REQ\_SCR\_DLINE, REQ\_SCR\_DPAGE, REQ\_SCR\_UPAGE sunt legate de derulare. Daca nu pot fi afisate toate articolele in sub-fereastra meniului atunci se poate spune despre meniu ca este "scrollable".? Aceste

cereri pot fi transmise functiei pentru a executa deplasarea cu o linie mai sus, mai jos sau cu o pagina mai sus respectiv jos.

- *REQ\_FIRST\_ITEM, REQ\_LAST\_ITEM, REQ\_NEXT\_ITEM si REQ\_PREV\_ITEM*

Aceste cereri sunt intuitive.

- *REQ\_TOGGLE\_ITEM*

Cand este transmisa aceasta cerere , ea prevede selectia prezenta. Optiunea poate fi folosita numai intr-un meniu cu mai multe valori?.Deci optiunea O\_ONEVALUE trebuie sa nu fie activata. Aceasta optiune poate fi activata sau dezactivata cu functia set\_menu\_opts().

- *Cereri legate de sabloane(Pattern requests)*

Fiecare meniu are asociat un buffer pentru sabloane (pattern buffer), care este folosit pentru a gasi cea mai apropiata potrivire pentru caracterele ascii introduse de utilizator. In orice moment in care sunt transmise caracterele ascii functiei menu\_driver(), ea le introduce in buffer-ul sablonului. Totodata incearca sa gaseasca cea mai apropiata potrivire conform sablonului in lista de articole si muta selectia curenta la articolul in cauza. Cererea REQ\_CLEAR\_PATTERN goleste buffer-ul sablonului. Cererea REQ\_BACK\_PATTERN sterge caracterul anterior in buffer-ul sablonului. In cazul in care sablonul se potriveste cu mai mult de un articol atunci ne putem deplasa prin articolele filtrate de sablon folosind REQ\_NEXT\_MATCH si REQ\_PREVIOUS\_MATCH .

- *Cereri ale mouse-ului*

In cazul cererilor KEY\_MOUSE, se executa o actiune in concordanta de pozitia mouse-ului. Actiunea care va avea loc este explicata in pagina manualului de utilizare (man) in felul urmator :

*Daca cel de-al doilea argument este tasta speciala KEY\_MOUSE, atunci evenimentul mouse asociat este translatat intr-una din cererile predefinite de mai sus.*

*La momentul curent numai click-urile in fereastra utilizator(ex. in cadrul zonei de expunere a meniului sau a decorarii ferestrei) sunt tratate. In momentul in care dati click deasupra zonei de afisare a meniului este generat un REQ\_SCR\_ULINE, daca dati dublu click este generat un mesaj de tip REQ\_SCR\_UPAGE iar daca dati click de trei ori este generat un REQ\_FIRST\_ITEM. Daca dati click dedesuptul zonei de afisare a meniului este generat un REQ\_SCR\_DLINE, daca dati dublu click este generat un REQ\_SCR\_DPAGE ,daca dati click de trei ori este generat un REQ\_LAST\_ITEM .Daca dati click pe un articol din cadrul zonei de afisare a meniului, cursorul meniului este positionat pe acel articol.*

Fiecare dintre cererile de deasupra vor fi explicate in randurile ce urmeaza cu cateva exemple cand e cazul.

## Ferestrele Meniului

Fiecarui meniu creat ii este asociat o fereastră si o subfereastră. Fereastră meniului afiseaza orice titlu sau bordura asociata meniului. Subfereastră meniului afiseaza articolele meniului care sunt disponibile pentru selectie la momentul curent. Dar noi n-am specificat vreo fereastră sau subfereastră in exemplul simplu. Cand nu este specificata o fereastră, stdscr este considerata fereastră principala, apoi meniul sistemului calculeaza marimea necesara subferestrei pentru a afisarea articolelor. Apoi articolele sunt afisate in sunfereastră calculata. Haideti sa ne jucam cu aceste ferestre si sa afisam un meniu cu o bordura si un titlu.

## Exemplu de ferestre ale meniului

```
/* File Path: menus/menu_win.c */
#include <menu.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD 4

char *choices[] = {
    "Choice 1",
    "Choice 2",
    "Choice 3",
    "Choice 4",
    "Exit",
    (char *)NULL,
};

void print_in_middle(WINDOW *win, int starty, int startx, int width,
char *string, chtype color);

int main()
{
    ITEM **my_items;
    int c;
    MENU *my_menu;
    WINDOW *my_menu_win;
    int n_choices, i;

    /* Initialize curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);
    init_pair(1, COLOR_RED, COLOR_BLACK);

    /* Create items */
    n_choices = ARRAY_SIZE(choices);
    my_items = (ITEM **)calloc(n_choices, sizeof(ITEM *));
    for(i = 0; i < n_choices; ++i)
        my_items[i] = new_item(choices[i], choices[i]);

    /* Create menu */
```

```

my_menu = new_menu((ITEM **)my_items);

/* Create the window to be associated with the menu */
my_menu_win = newwin(10, 40, 4, 4);
keypad(my_menu_win, TRUE);

/* Set main window and sub window */
set_menu_win(my_menu, my_menu_win);
set_menu_sub(my_menu, derwin(my_menu_win, 6, 38, 3, 1));

/* Set menu mark to the string " * " */
set_menu_mark(my_menu, " * ");

/* Print a border around the main window and print a title */
box(my_menu_win, 0, 0);
print_in_middle(my_menu_win, 1, 0, 40, "My Menu", COLOR_PAIR(1));
mvwaddch(my_menu_win, 2, 0, ACS_LTEE);
mvwhline(my_menu_win, 2, 1, ACS_HLINE, 38);
mvwaddch(my_menu_win, 2, 39, ACS_RTEE);

/* Post the menu */
post_menu(my_menu);
wrefresh(my_menu_win);

while((c = wgetch(my_menu_win)) != KEY_F(1))
{
    switch(c)
    {
        case KEY_DOWN:
            menu_driver(my_menu, REQ_DOWN_ITEM);
            break;
        case KEY_UP:
            menu_driver(my_menu, REQ_UP_ITEM);
            break;
    }
    wrefresh(my_menu_win);
}

/* Unpost and free all the memory taken up */
unpost_menu(my_menu);
free_menu(my_menu);
for(i = 0; i < n_choices; ++i)
    free_item(my_items[i]);
endwin();
}

void print_in_middle(WINDOW *win, int starty, int startx, int width,
char *string, chtype color)
{
    int length, x, y;
    float temp;

    if(win == NULL)
        win = stdscr;
    getyx(win, y, x);
    if(startx != 0)
        x = startx;
    if(starty != 0)
        y = starty;
    if(width == 0)

```

```

        width = 80;

        length = strlen(string);
        temp = (width - length)/ 2;
        x = startx + (int)temp;
        wattron(win, color);
        mvwprintw(win, y, x, "%s", string);
        wattroff(win, color);
        refresh();
    }

```

In acest exemplu am creat un meniu cu un titlu, bordura, o linie eleganta care separa titlul si articolele. Dupa cum puteti vedea, pentru a atasa o fereastră unui meniu trebuie utilizata functia `set_menu_win()`. Apoi atasam si subfereastră. Acest lucru realizeaza afisarea articolelor in subfereastră. Puteti de asemenea stabili string-ul marcat care este afisat la stanga articolului selectat cu `set_menu_mark()`.

## Meniuri derulabile

Daca subfereastră nu este suficient de mare pentru a expune toate articolele atunci meniul va fi derulant. Cand sunteti positionati pe ultimul articol in lista prezenta si trimiteti `REQ_DOWN_ITEM`, este transformat in `REQ_SCR_DLINE` iar meniul se deruleaza cu cate un articol. Puteti transmite manual `REQ_SCR_operations` pentru derulare. Dar sa vedem cum se poate face .

## Exemplu de meniuri derulabile

```

/* File Path: menus/menu_scroll.c */
#include <menu.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD 4

char *choices[] = {
    "Choice 1",
    "Choice 2",
    "Choice 3",
    "Choice 4",
    "Choice 5",
    "Choice 6",
    "Choice 7",
    "Choice 8",
    "Choice 9",
    "Choice 10",
    "Exit",
    (char *)NULL,
};

void print_in_middle(WINDOW *win, int starty, int startx, int width,
char *string, chtype color);

int main()
{
    ITEM **my_items;
    int c;

```

```

MENU *my_menu;
WINDOW *my_menu_win;
int n_choices, i;

/* Initialize curses */
initscr();
start_color();
cbreak();
noecho();
keypad(stdscr, TRUE);
init_pair(1, COLOR_RED, COLOR_BLACK);
init_pair(2, COLOR_CYAN, COLOR_BLACK);

/* Create items */
n_choices = ARRAY_SIZE(choices);
my_items = (ITEM **)calloc(n_choices, sizeof(ITEM *));
for(i = 0; i < n_choices; ++i)
    my_items[i] = new_item(choices[i], choices[i]);

/* Create menu */
my_menu = new_menu((ITEM **)my_items);

/* Create the window to be associated with the menu */
my_menu_win = newwin(10, 40, 4, 4);
keypad(my_menu_win, TRUE);

/* Set main window and sub window */
set_menu_win(my_menu, my_menu_win);
set_menu_sub(my_menu, derwin(my_menu_win, 6, 38, 3, 1));
set_menu_format(my_menu, 5, 1);

/* Set menu mark to the string " * " */
set_menu_mark(my_menu, " * ");

/* Print a border around the main window and print a title */
box(my_menu_win, 0, 0);
print_in_middle(my_menu_win, 1, 0, 40, "My Menu", COLOR_PAIR(1));
mvwaddch(my_menu_win, 2, 0, ACS_LTEE);
mvwhline(my_menu_win, 2, 1, ACS_HLINE, 38);
mvwaddch(my_menu_win, 2, 39, ACS_RTEE);

/* Post the menu */
post_menu(my_menu);
wrefresh(my_menu_win);

attron(COLOR_PAIR(2));
mvprintw(LINES - 2, 0, "Use PageUp and PageDown to scoll up or
down a page of items");
mvprintw(LINES - 1, 0, "Arrow Keys to navigate (F1 to Exit)");
attroff(COLOR_PAIR(2));
refresh();

while((c = wgetch(my_menu_win)) != KEY_F(1))
{
    switch(c)
    {
        case KEY_DOWN:
            menu_driver(my_menu, REQ_DOWN_ITEM);
            break;

```

```

        case KEY_UP:
            menu_driver(my_menu, REQ_UP_ITEM);
            break;
        case KEY_NPAGE: /* Page Down */
            menu_driver(my_menu, REQ_SCR_DPAGE);
            break;
        case KEY_PPAGE: /* Page Up */
            menu_driver(my_menu, REQ_SCR_UPAGE);
            break;
    }
    wrefresh(my_menu_win);
}

/* Unpost and free all the memory taken up */
unpost_menu(my_menu);
free_menu(my_menu);
for(i = 0; i < n_choices; ++i)
    free_item(my_items[i]);
endwin();
}

void print_in_middle(WINDOW *win, int starty, int startx, int width,
char *string, chtype color)
{
    int length, x, y;
    float temp;

    if(win == NULL)
        win = stdscr;
    getyx(win, y, x);
    if(startx != 0)
        x = startx;
    if(starty != 0)
        y = starty;
    if(width == 0)
        width = 80;

    length = strlen(string);
    temp = (width - length) / 2;
    x = startx + (int)temp;
    wattron(win, color);
    mvwprintw(win, y, x, "%s", string);
    wattroff(win, color);
    refresh();
}

```

Programul se explica de la sine. In acest exemplu numarul optiunilor a fost crescut la 10, ceea ce este mai mult decat poate afisa zona subferestrei, in speța 6 articole. Acest mesaj trebuie transmis explicit meniului sistem cu functia `set_menu_format()`. Trebuie specificat numarul de coloane si linii care dorim sa fie afisate intr-o singura pagina. Putem specifica orice numar privitor la articolele care vrem sa fie afisate atata timp cat numarul este mai mic decat inaltimea subferestrei. Daca tasta apasata de utilizator este PAGE UP sau PAGE DOWN, meniul este derulat cu o pagina datorita cererilor (`REQ_SCR_DPAGE` si `REQ_SCR_UPAGE`) transmise functiei `menu_driver()`.

## Meniuri cu mai multe coloane

In exemplul de mai sus ati vazut cum modul de utilizare a functiei `set_menu_format()`. Nu am mentionat ce face variabila `cols`(al treilea parametru). Ei bine, daca subfereastra e suficient de lata, puteti opta sa afisati mai mult de un articol pe o linie. Acest lucru poate fi specificat in variabila `cols`. Pentru a simplifica lucrurile, urmatorul exemplu nu afiseaza descrieri pentru articole.

## Exemplu de meniu cu mai multe coloane

```
/* File Path: menus/menu_multi_column.c */
#include <menu.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD 4

char *choices[] = {
    "Choice 1", "Choice 2", "Choice 3", "Choice 4",
    "Choice 5",
    "Choice 6", "Choice 7", "Choice 8", "Choice 9",
    "Choice 10",
    "Choice 11", "Choice 12", "Choice 13", "Choice
14", "Choice 15",
    "Choice 16", "Choice 17", "Choice 18", "Choice
19", "Choice 20",
    "Exit",
    (char *)NULL,
};

int main()
{
    ITEM **my_items;
    int c;
    MENU *my_menu;
    WINDOW *my_menu_win;
    int n_choices, i;

    /* Initialize curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);
    init_pair(1, COLOR_RED, COLOR_BLACK);
    init_pair(2, COLOR_CYAN, COLOR_BLACK);

    /* Create items */
    n_choices = ARRAY_SIZE(choices);
    my_items = (ITEM **)calloc(n_choices, sizeof(ITEM *));
    for(i = 0; i < n_choices; ++i)
        my_items[i] = new_item(choices[i], choices[i]);

    /* Create menu */
    my_menu = new_menu((ITEM **)my_items);
```

```

/* Set menu option not to show the description */
menu_opts_off(my_menu, O_SHOWDESC);

/* Create the window to be associated with the menu */
my_menu_win = newwin(10, 70, 4, 4);
keypad(my_menu_win, TRUE);

/* Set main window and sub window */
set_menu_win(my_menu, my_menu_win);
set_menu_sub(my_menu, derwin(my_menu_win, 6, 68, 3, 1));
set_menu_format(my_menu, 5, 3);
set_menu_mark(my_menu, " * ");

/* Print a border around the main window and print a title */
box(my_menu_win, 0, 0);

attron(COLOR_PAIR(2));
mvprintw(LINES - 3, 0, "Use PageUp and PageDown to scroll");
mvprintw(LINES - 2, 0, "Use Arrow Keys to navigate (F1 to
Exit)");
attroff(COLOR_PAIR(2));
refresh();

/* Post the menu */
post_menu(my_menu);
wrefresh(my_menu_win);

while((c = wgetch(my_menu_win)) != KEY_F(1))
{
    switch(c)
    {
        case KEY_DOWN:
            menu_driver(my_menu, REQ_DOWN_ITEM);
            break;
        case KEY_UP:
            menu_driver(my_menu, REQ_UP_ITEM);
            break;
        case KEY_LEFT:
            menu_driver(my_menu, REQ_LEFT_ITEM);
            break;
        case KEY_RIGHT:
            menu_driver(my_menu, REQ_RIGHT_ITEM);
            break;
        case KEY_NPAGE:
            menu_driver(my_menu, REQ_SCR_DPAGE);
            break;
        case KEY_PPAGE:
            menu_driver(my_menu, REQ_SCR_UPAGE);
            break;
    }
    wrefresh(my_menu_win);
}

/* Unpost and free all the memory taken up */
unpost_menu(my_menu);
free_menu(my_menu);
for(i = 0; i < n_choices; ++i)
    free_item(my_items[i]);
endwin();

```

```
}
```

Urmăriti apelul funcției `set_menu_format()`. Specifica ca numărul coloanelor să fie 3, deci afișează 3 articole pe linie. De asemenea am dezactivat descrierile de afișare cu ajutorul funcției `menu_opt_off()`. Sunt câteva funcții `set_menu_opts()`, `menu_opts_on()` și `menu_opts_off()` care pot fi utilizate pentru a manipula opțiunile meniului. Pot fi specificate următoarele opțiuni ale meniului.

`O_ONEVALUE`

Numai un singur articol poate fi selectat pentru acest meniu

`O_SHOWDESC`

Afișează descrierile articolului când meniul este postat.

`O_ROWMAJOR`

Afișează meniul în ordinea row-major.

`O_IGNORECASE`

Ignoră cazul în cazul în care există o potrivire în șablonul de căutare

`O_SHOWMATCH`

Muta cursorul pe numele articolului care a fost găsit în urma căutării.

`O_NONCYCLIC`

Nu leagă următorul item de cel dinaintea, cerând alt sfârșit de meniu

Toate opțiunile sunt implicit setate activ. Puteti activa sau dezactiva atribute specifice cu funcțiile `menu_opts_on()` și `menu_opts_off()`. Puteti de asemenea utiliza `set_menu_opts()` pentru a specifica direct opțiunile. Argumentul acestei funcții ar trebui să fie o valoare obținută din câteva din constantele de mai sus legate prin OR. Funcția `menu_opts()` poate fi folosită pentru a afla opțiunile actuale ale meniului.

## Meniuri multivaluate

Poate vă întrebați ce-ar fi dacă ați dezactiva opțiunea `O_ONEVALUE`. În acest caz meniul devine multivaluat. Asta înseamnă că puteți selecta mai mult decât un articol. Acest lucru ne aduce în atenție funcția `REQ_TOGGLE_ITEM`. Să o vedem în acțiune.

## Exemplu de meniuri multivaluate

```
/* File Path: menus/menu_toggle.c */
#include <menu.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD 4

char *choices[] = {
    "Choice 1",
    "Choice 2",
    "Choice 3",
    "Choice 4",
    "Choice 5",
    "Choice 6",
    "Choice 7",
    "Exit",
};

int main()
{
    ITEM **my_items;
    int c;
    MENU *my_menu;
    int n_choices, i;
    ITEM *cur_item;

    /* Initialize curses */
    initscr();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* Initialize items */
    n_choices = ARRAY_SIZE(choices);
    my_items = (ITEM **)calloc(n_choices + 1, sizeof(ITEM *));
    for(i = 0; i < n_choices; ++i)
        my_items[i] = new_item(choices[i], choices[i]);
    my_items[n_choices] = (ITEM *)NULL;

    my_menu = new_menu((ITEM **)my_items);

    /* Make the menu multi valued */
    menu_opts_off(my_menu, O_ONEVALUE);

    mvprintw(LINES - 3, 0, "Use <SPACE> to select or unselect an
item.");
    mvprintw(LINES - 2, 0, "<ENTER> to see presently selected items(F1
to Exit)");
    post_menu(my_menu);
    refresh();

    while((c = getch()) != KEY_F(1))
    {
        switch(c)
        {
            case KEY_DOWN:
                menu_driver(my_menu, REQ_DOWN_ITEM);
                break;
        }
    }
}
```

```

        case KEY_UP:
            menu_driver(my_menu, REQ_UP_ITEM);
            break;
        case ' ':
            menu_driver(my_menu, REQ_TOGGLE_ITEM);
            break;
        case 10: /* Enter */
        {
            char temp[200];
            ITEM **items;

            items = menu_items(my_menu);
            temp[0] = '\0';
            for(i = 0; i < item_count(my_menu); ++i)
                if(item_value(items[i]) == TRUE)
                    {
                        strcat(temp,
                                item_name(items[i]));
                                strcat(temp, " ");
                    }
            move(20, 0);
            clrtoeol();
            mvprintw(20, 0, temp);
            refresh();
        }
        break;
    }
}
unpost_menu(my_menu);
for(i = 0; i < n_choices; ++i)
    free_item(my_items[i]);
free_menu(my_menu);
endwin();
}

```

Uau. O multime de functii noi. Sa le luam pe rand. In primul rand REQ\_TOGGLE\_ITEM. Intr-un meniu multivaluat utilizatorului ar trebui sa i se permita selectia sau deselectia a mai mult decat un articol. Cererea REQ\_TOGGLE\_ITEM prevede selectia prezenta. In cazul de fata cand este apasat space functiei menu\_driver ii este trimisa cererea REQ\_TOGGLE\_ITEM pentru a obtine rezultatul..

Acum, cand utilizatorul apasa <ENTER> vom afisa atricolele pe care sunt selectate in momentul curent. Mai intai aflam care sunt articolele asociate meniului folosind functia menu\_items(). Apoi buclam prin articole pentru a afla daca un articol este selectat sau nu. Functia item\_value() intoarce TRUE daca un articol este selectat. Functia item\_count() intoarce numarul de articole ale meniului. Numele articolului poate fi aflat cu functia item\_name(). Puteti gasi descrierea asociata unui articol folosind item\_descriotion().

## Optiuni ale meniului

Ei bine, probabil ca de acum simtiti nevoia unei schimbari in meniul vostru, cu multa functionalitate. Stiu. Vreti Culori !!! Vreti sa creati meniuri dragute similare cu acele [jocuri dos](#) in mod text. Functiile set\_menu\_fore() si set\_menu\_black() pot fi folosite pentru a schimba atributul articolului selectat si al articolului deselectat. Numele sunt inselatoare. Ele nu schimba

foreground-ul sau background-ul meniului, ceea ce ar fi fost fara folos. Functia `set_menu_grey()` poate fi folosita pentru a seta atributul de afisare pentru articolele meniului ce nu pot fi selectate. Acest lucru ne aduce in atentie interesanta optiune pentru un aricol : `O_SELECTABLE`. O putem anula folosind functia `item_opts_off()` iar apoi articolul nu mai poate fi selectat. Este ca un articol gri din elegantele meniuri Windows. Sa punem aceste concepte in practica cu urmatorul exemplu.

## Exemplu despre Optiunile de Meniu

```
/* File Path: menus/menu_attrib.c */
#include <menu.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD 4

char *choices[] = {
    "Choice 1",
    "Choice 2",
    "Choice 3",
    "Choice 4",
    "Choice 5",
    "Choice 6",
    "Choice 7",
    "Exit",
};

int main()
{
    ITEM **my_items;
    int c;
    MENU *my_menu;
    int n_choices, i;
    ITEM *cur_item;

    /* Initialize curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);
    init_pair(1, COLOR_RED, COLOR_BLACK);
    init_pair(2, COLOR_GREEN, COLOR_BLACK);
    init_pair(3, COLOR_MAGENTA, COLOR_BLACK);

    /* Initialize items */
    n_choices = ARRAY_SIZE(choices);
    my_items = (ITEM **)calloc(n_choices + 1, sizeof(ITEM *));
    for(i = 0; i < n_choices; ++i)
        my_items[i] = new_item(choices[i], choices[i]);
    my_items[n_choices] = (ITEM *)NULL;
    item_opts_off(my_items[3], O_SELECTABLE);
    item_opts_off(my_items[6], O_SELECTABLE);

    /* Create menu */
    my_menu = new_menu((ITEM **)my_items);
```

```

    /* Set fore ground and back ground of the menu */
    set_menu_fore(my_menu, COLOR_PAIR(1) | A_REVERSE);
    set_menu_back(my_menu, COLOR_PAIR(2));
    set_menu_grey(my_menu, COLOR_PAIR(3));

    /* Post the menu */
    mvprintw(LINES - 3, 0, "Press <ENTER> to see the option selected");
    mvprintw(LINES - 2, 0, "Up and Down arrow keys to naviage (F1 to
Exit)");
    post_menu(my_menu);
    refresh();

    while((c = getch()) != KEY_F(1))
    {
        switch(c)
        {
            case KEY_DOWN:
                menu_driver(my_menu, REQ_DOWN_ITEM);
                break;
            case KEY_UP:
                menu_driver(my_menu, REQ_UP_ITEM);
                break;
            case 10: /* Enter */
                move(20, 0);
                clrtoeol();
                mvprintw(20, 0, "Item selected is : %s",

item_name(current_item(my_menu)));
                pos_menu_cursor(my_menu);
                break;

        }
    }
    unpost_menu(my_menu);
    for(i = 0; i < n_choices; ++i)
        free_item(my_items[i]);
    free_menu(my_menu);
    endwin();
}

```

## Utilitatea pointerilor utilizator

Putem asocia un pointer utilizator cu fiecare articol al meniului. Lucreaza la fel ca un pointer utilizator in panel-uri. Nu este atins de meniul sistem.Puteti stoca orice doriti in el.De obicei eu il folosesc pentru a stoca functia ce va fi executata cand optiunea meniului este aleasa. (Este selectata si poate utilizatorul a apasat <ENTER>);

## Exemplu pentru pointerul utilizator

```

/* File Path: menus/menu_userptr.c */
#include <menu.h>

#define ARRAY_SIZE(a) (sizeof(a) / sizeof(a[0]))
#define CTRLD 4

```

```

char *choices[] = {
    "Choice 1",
    "Choice 2",
    "Choice 3",
    "Choice 4",
    "Choice 5",
    "Choice 6",
    "Choice 7",
    "Exit",
};
void func(char *name);

int main()
{
    ITEM **my_items;
    int c;
    MENU *my_menu;
    int n_choices, i;
    ITEM *cur_item;

    /* Initialize curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);
    init_pair(1, COLOR_RED, COLOR_BLACK);
    init_pair(2, COLOR_GREEN, COLOR_BLACK);
    init_pair(3, COLOR_MAGENTA, COLOR_BLACK);

    /* Initialize items */
    n_choices = ARRAY_SIZE(choices);
    my_items = (ITEM **)calloc(n_choices + 1, sizeof(ITEM *));
    for(i = 0; i < n_choices; ++i)
    {
        my_items[i] = new_item(choices[i], choices[i]);
        /* Set the user pointer */
        set_item_userptr(my_items[i], func);
    }
    my_items[n_choices] = (ITEM *)NULL;

    /* Create menu */
    my_menu = new_menu((ITEM **)my_items);

    /* Post the menu */
    mvprintw(LINES - 3, 0, "Press <ENTER> to see the option selected");
    mvprintw(LINES - 2, 0, "Up and Down arrow keys to naviage (F1 to
Exit)");
    post_menu(my_menu);
    refresh();

    while((c = getch()) != KEY_F(1))
    {
        switch(c)
        {
            case KEY_DOWN:
                menu_driver(my_menu, REQ_DOWN_ITEM);
                break;
            case KEY_UP:

```

```

        menu_driver(my_menu, REQ_UP_ITEM);
        break;
    case 10: /* Enter */
    {
        ITEM *cur;
        void (*p)(char *);

        cur = current_item(my_menu);
        p = item_userptr(cur);
        p((char *)item_name(cur));
        pos_menu_cursor(my_menu);
        break;
    }
    break;
}
}
unpost_menu(my_menu);
for(i = 0; i < n_choices; ++i)
    free_item(my_items[i]);
free_menu(my_menu);
endwin();
}

void func(char *name)
{
    move(20, 0);
    clrtoeol();
    mvprintw(20, 0, "Item selected is : %s", name);
}

```

## Biblioteca pentru form-uri

Ei bine, daca ati vazut acele form-uri pe paginile web care iau datele de intrare de la utilizatori si fac diferite lucruri, probabil va intrebati cum poate cineva sa creeze asemenea form-uri in afisare in mod text. Este cam dificil sa scriem acele forme dichisite in ncurses pur. Biblioteca de form-uri incearca sa furnizeze un cadru de lucru de baza pentru a construi si a mentine form-urile cu usurinta. Are multe functii care se ocupa de validare, extinderea dinamica a campurilor etc.. Sa-o vedem la lucru.

Un form este o colectie de campuri; fiecare camp poate fie o eticheta(text static) sau o locatie de intrare a datelor. De asemenea biblioteca de form-uri ofera functii pentru a diviza form-urile in mai multe pagini.

### Lucruri de baza

Form-rile sunt create cam in aceeasi masura ca si meniurile. Mai intai sunt create campurile ce au legatura cu form-ul folosind `new_field()`. Muteti fixa diferite optiuni pentru campuri, asa incat sa fie afisate cu tot felul de attribute particularizate, validate inaintea pierderii focuului de catre camp etc. Apoi campurile sunt atasate form-ului. Apoi form-ul este afisat si pregatit sa preia date de intrare.La fel ca la `menu_driver()`, form-ul este manipulat de `form_driver()`. Putem trimite cereri catre `form_driver` pentru a muta focus-ul pe un anumit camp, a muta

cursor-ul la capatul campului etc. . Dupa ce utilizatorul a introdus valorile si validarea e terminata, form-ul poate fi luat de pe ecran si memoria dealocata.

In general transmiterea controlului in cazul unui program ce foloseste form-uri arata cam asa:

1. Initializare curses
2. Creare de campuri folosind `new_field()`. Puteti specifica inaltimea si latimea campului, si pozitia acestuia in form.
3. Creati form-urile cu `new_form()` specificand campurile care vor fi atasate.
4. postati form-ul cu `form_post()` si dati refresh ecranului.
5. Prelucrati cererile utilizatorului intr-o bucla si faceti modificarile necesare form-ului utilizand `form_driver`.
6. Luati form-ul de pe ecran cu `form_unpost()`
7. Eliberati memoria alocata form-ului cu `free_form()`
8. Eliberati memoria alocata articolelor cu `free_field()`
9. Terminati curses

Dupa cum puteti vedea, lucrand cu biblioteca pentru form-uri e in mare masura similar cu manevrarea bibliotecii pentru meniu. In exemplele ce urmeaza vom explora aspecte diferite ale prelucrarii form-urilor. Pentru inceput sa ne incepem calatoria cu un exemplu simplu.

## Compilarea folosind biblioteca pentru form-uri

Pentru a utiliza functii din biblioteca form-urilor trebuie sa includem `form.h` iar pentru a face legatura intre program si biblioteca pentru form-uri flag-ul `-lform` trebuie adaugat impreuna cu `-lcurses` in ordinea precizata.

```
#include <form.h>
.
.
.

compilare si link-editare: gcc <fisier program> -lform -lcurses
```

## Un exemplu simplu pentru folosirea bibliotecii de form-uri

```
/* File Path: forms/form_simple.c */
#include <form.h>

int main()
{
    FIELD *field[3];
    FORM *my_form;
    int ch;

    /* Initialize curses */
    initscr();
    cbreak();
```

```

noecho();
keypad(stdscr, TRUE);

/* Initialize the fields */
field[0] = new_field(1, 10, 4, 18, 0, 0);
field[1] = new_field(1, 10, 6, 18, 0, 0);
field[2] = NULL;

/* Set field options */
set_field_back(field[0], A_UNDERLINE);          /* Print a line for the
option */
field_opts_off(field[0], O_AUTOSKIP);          /* Don't go to next
field when this */

/* Field is filled up
*/
set_field_back(field[1], A_UNDERLINE);
field_opts_off(field[1], O_AUTOSKIP);

/* Create the form and post it */
my_form = new_form(field);
post_form(my_form);
refresh();

mvprintw(4, 10, "Value 1:");
mvprintw(6, 10, "Value 2:");
refresh();

/* Loop through to get user requests */
while((ch = getch()) != KEY_F(1))
{
    switch(ch)
    {
        case KEY_DOWN:
            /* Go to next field */
            form_driver(my_form, REQ_NEXT_FIELD);
            /* Go to the end of the present buffer */
            /* Leaves nicely at the last character */
            form_driver(my_form, REQ_END_LINE);
            break;
        case KEY_UP:
            /* Go to previous field */
            form_driver(my_form, REQ_PREV_FIELD);
            form_driver(my_form, REQ_END_LINE);
            break;
        default:
            /* If this is a normal character, it gets */
            /* Printed */
            form_driver(my_form, ch);
            break;
    }
}

/* Un post form and free the memory */
unpost_form(my_form);
free_form(my_form);
free_field(field[0]);
free_field(field[1]);

```

```
endwin();
return 0;
}
```

Exmplul de mai sus este cat se poate de elocvent. Se creaza doua campuri cu `new_field()`. `new_field()` ia ca parametri `height`, `width`, `starty`, `startx`, numarul de coloane din afara ecranului si numarul bufferelor de lucru aditionale. Al cincilea argument, numarul de coloane din afara ecranului, precizeaza cat va fi afisat din camp. Daca este zero, atunci intregul camp este afisat intotdeauna altfel form-ul va fi derulabil atunci cand utilizatorul acceseaza parti ale campului care nu sunt expuse. Biblioteca form-urilor aloca un buffer pentru fiecare camp in scopul de a stoca datele de intrare ale utilizatorului. Folosind ultimul parametru al functiei `new_field()` putem aloca buffere aditionale. Acestea pot fi utilizate in orice scop doriti.

Dupa ce ati creat campurile, atributul `background` al ambelor este setat la underscore folosind `set_field_back()`. Optiunea `AUTOSKIP` este dezactivata folosind `field_opts_off()`. Daca aceasta optiune este dezactivata, focusul se va muta la urmatorul camp al form-ului de indata ce campul activ se va fi umplut complet.

Dupa ce ati atasat campurile form-ului, postam form-ul. De aici inainte, datele de intrare de la utilizatori sunt prelucrate in bucla `while`, facand cereri corespunzatoare catre `form_driver`. Detaliile tuturor cererilor catre `form_driver` sunt explicate mai taziu.

## Jocandu-ne cu campuri

Fiecare camp al form-ului e asociat cu o multime de attribute. Ele pot fi manipulate pentru a obtine efectul scontat si pentru a ne distra!!!. Asa ca de ce sa asteptam?

## Aflarea marimii si locatiei campului

Parametrii pe care i-am dat la crearea campului pot fi aflati cu `field_info()`. Ne intoarce `height`, `width`, `starty`, `startx`, numarul de coloane din afara ecranului si numarul bufferelor de lucru aditionale in parametrii pe care i-am dat functiei. Este un fel de inversa a lui `new_field()`.

```
int field_info(FIELD *field, /* campul din care luam datele */
int *height, *int width, /* campul size */
int *top, int *left, /* coltul stanga sus*/
int *offscreen, /*nr coloane din afara ecranului*/
int *nbuf); /*nr de buffere lucratoare */
```

## Deplasarea campului

Locatia campului poate fi chimbata la o alta pozitie cu `move_field()`.

```
int move_field(FIELD *field, /*campul de modificat */
int top, int left); /*noul colt stanga sus*/
```

Ca de obicei, pozitia schimbata poate fi interogata cu field\_infor().

## Alinierea campurilor

Putem sa realizam tipul de aliniere dorit pentru un camp folosind functia set\_field\_just().

```
int set_field_just(FIELD *field, /*campul ce trebuie modificat*/
int justmode); /*modul de setare */
int field_just(FIELD *field); /*reda modul de aliniere al campului*/
```

Modurile de aliniere acceptat si returnat de aceste functii sunt NO\_JUSTIFICATION, JUSTIFY\_RIGHT, JUSTIFY\_LEFT, sau JUSTIFY\_CENTER.

## Atribute ale afisarii campurilor

Asa cum ati vazut, in exemplul de mai sus, atributele de afisare pentru campuri pot fi setate cu ajutorul set\_field\_fore() si set\_field\_back(). Aceste functii seteaza atributele foreground-ului si ale background-ului. Puteti de asemenea specifica un sir de caractere de umplutura cu care vom umple zona campului care nu a fost completata. Caracterul de umplutura poate fi setat cu un apel al functiei set\_field\_pad(). Valoarea implicita este spatiu. Puteti utiliza functiile field\_fore(), field\_back, field\_pad() pentru a interoga atributele curente ale foregroundului, backgroundului si caracterul "de umplutura" pentru campul in cauza. In lista urmatoare In continuare aveti o lista cu modul de utilizare al functiilor :

```
int set_field_fore(FIELD *field, /*camp ce trebuie modificat */
chtype attr); /*atributul ce trebuie setat*/

chtype field_fore(FIELD *field); /*camp ce trebuie interogat */
/*intoarce atributul foregroundului */

int set_field_back(FIELD *field, /*camp ce trebuie modificat */
chtype attr); /*atributul ce trebuie setat */

chtype field_back(FIELD *field); /* camp ce trebuie interogat */
/*intoarce atributul backgroundului */

int set_field_pad(FIELD *field, /* camp ce trebuie modificat */
int pad); /*caracter de "umplutura"*/

chtype field_pad(FIELD *field); /*camp ce trebuie interogat*/
/*intoarce caracterul de "umplutura"
actual*/
```

Chiar daca functiile de mai sus par simple, poate fi frustrant la inceput folosirea de culori cu set\_field\_fore(). Mai intai sa explic despre atributele foregroundului si ale backgroundului unui camp. Atributul foregroundului este asociat cu carecterul. Ceea ce inseamna ca in camp este

afisat un caracter laolalta cu atributul ce l-ati setat cu `set_field_fore()`. Atributul `background`ului este cel utilizat pentru a umple backgroundul campului, chiar daca acolo este vreun caracter sau nu. Cum ramane cu culorile ? Deoarece culorile sunt intotdeauna definite in perechi, care este metoda corecta de a afisa campuri colorate ? Aici este un exemplu pentru clarificarea atributelor culoilor

## Exemplu pentru attributele afisarii

```
/* File Path: forms/form_attrib.c */
#include <form.h>

int main()
{
    FIELD *field[3];
    FORM *my_form;
    int ch;

    /* Initialize curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* Initialize few color pairs */
    init_pair(1, COLOR_WHITE, COLOR_BLUE);
    init_pair(2, COLOR_WHITE, COLOR_BLUE);

    /* Initialize the fields */
    field[0] = new_field(1, 10, 4, 18, 0, 0);
    field[1] = new_field(1, 10, 6, 18, 0, 0);
    field[2] = NULL;

    /* Set field options */
    set_field_fore(field[0], COLOR_PAIR(1)); /* Put the field with blue
background */
    set_field_back(field[0], COLOR_PAIR(2)); /* and white foreground
(characters */
  /* are printed in white
*/
    field_opts_off(field[0], O_AUTOSKIP); /* Don't go to next
field when this */
  /* Field is filled up
*/
    set_field_back(field[1], A_UNDERLINE);
    field_opts_off(field[1], O_AUTOSKIP);

    /* Create the form and post it */
    my_form = new_form(field);
    post_form(my_form);
    refresh();

    set_current_field(my_form, field[0]); /* Set focus to the colored
field */
    mvprintw(4, 10, "Value 1:");
    mvprintw(6, 10, "Value 2:");
}
```

```

        mvprintw(LINES - 2, 0, "Use UP, DOWN arrow keys to switch between
fields");
        refresh();

        /* Loop through to get user requests */
        while((ch = getch()) != KEY_F(1))
        {
            switch(ch)
            {
                case KEY_DOWN:
                    /* Go to next field */
                    form_driver(my_form, REQ_NEXT_FIELD);
                    /* Go to the end of the present buffer */
                    /* Leaves nicely at the last character */
                    form_driver(my_form, REQ_END_LINE);
                    break;
                case KEY_UP:
                    /* Go to previous field */
                    form_driver(my_form, REQ_PREV_FIELD);
                    form_driver(my_form, REQ_END_LINE);
                    break;
                default:
                    /* If this is a normal character, it gets */
                    /* Printed */
                    form_driver(my_form, ch);
                    break;
            }
        }

        /* Un post form and free the memory */
        unpost_form(my_form);
        free_form(my_form);
        free_field(field[0]);
        free_field(field[1]);

        endwin();
        return 0;
}

```

Jucati-va cu perechile de culori si incercati sa intelegeti atributetele foregroundului si ale backgroundului. In programele in care folosesc atributetele culorilor setez doar backgroundul cu `set_field_back()`. Pur si simplu curses nu permite definirea atributetele pentru culori individuale.

## Bitii pentru optiunile campului

Exista o colectie mare de biti ai optiunilor unui camp pe care ii puteti seta pentru a controla aspecte variate ale prelucrarii formurilor. Ii puteti manipula cu aceste functii:

```

int set_field_opts(FIELD *field,          /*camp de modificat*/
                  int attr);             /* atribut ce trebuie setat */

int field_opts_on(FIELD *field,          /*camp de modificat*/
                 int attr);             /*atribute ce trebuie activate*/

int field_opts_off(FIELD *field,         /*camp de modificat*/

```

```
int attr);          /*atribute ce trebuiesc deselectate*/  
  
int field_opts(FIELD *field);          /*camp ce trebuie interogat*/
```

Funcția `set_field_opts()` poate fi folosită pentru a seta direct atributele unui câmp sau puteți alege să selectați sau să deselectați câteva atribute cu utilizând `selective field_opts_on()` și `field_opts_off()`. Puteți interoga oricând atributele unui câmp cu `field_opts()`. În continuare avem lista cu opțiuni valabile. Implicit toate opțiunile sunt selectate.

#### O\_VISIBLE

Controlează vizibilitatea câmpului pe ecran. Poate fi utilizată în timpul prelucrării formurilor în scopul ascunderii sau apariției formurilor în funcție de valoarea câmpurilor părinte.

#### O\_ACTIVE

Controlează dacă un câmp este activ în timpul prelucrării formurilor (ex. fiind vizitat de tastele de navigare ale formularului). Poate fi utilizat pentru a face etichete sau câmpuri derivate cu valori ale bufferului ce pot fi alterate de formularele aplicației și nu de utilizator.

#### O\_PUBLIC

Controlează dacă datele sunt afișate în timp ce sunt introduse în câmp. Dacă această opțiune este selectată într-un câmp, biblioteca va accepta orice dată editabilă în acel câmp, dar nu va fi afișată și cursorul vizibil din cadrul câmpului nu se va mișca. Puteți deselecta bitul `O_PUBLIC` pentru a defini câmpuri parolă.

#### O\_EDIT

Controlează dacă datele câmpului pot fi modificate. Când această opțiune nu este activată, toate cererile de activare cu excepția `REQ_PREV_CHOICE` și `REQ_NEXT_CHOICE` vor esua. Aceste câmpuri de tip `read-only` poate fi folosite pentru mesaje de ajutor.

#### O\_WRAP

Controlează organizarea cuvintelor în câmpuri de tip multilinie. În mod normal, când orice caracter al unui cuvânt (separat de spațiu) ajunge la capătul liniei curente, întregul cuvânt este trimis liniei următoare (presupunând că există una). Când această opțiune nu este activată, cuvântul va fi împărțit după sfârșitul liniei.

#### O\_BLANK

Controlează golirea câmpurilor. Când această opțiune este setată, dacă introducem un caracter pe prima poziție din câmp se șterge conținutul întregului câmp (excepție făcând caracterul abia introdus).

#### O\_AUTOSKIP

Controlează trecerea automată la următorul câmp când cel actual se umple. În mod normal, când utilizatorul formurilor încearcă să tiparească mai multe date decât într-un câmp, locația de editare trece la următorul câmp. Când această opțiune este dezactivată, cursorul utilizatorului va zăbovi la capătul câmpului. Această opțiune este ignorată în câmpurile dinamice care nu și-au atins limita.

#### O\_NULLOK

Controlează dacă validarea este aplicată câmpurilor goale. În mod normal nu se aplică; utilizatorul poate lăsa un câmp gol fără a invoca validarea usuală la ieșire. Dacă această opțiune este inactivă într-un câmp, parasirea câmpului invocă o validare.

#### O\_PASSOK

Controleaza daca validarea are loc la fiecare iesire, sau numai dupa ce campul este modificat. In mod normal ultima varianta e adevarata. Setarea lui O\_PASSOK poate folosi daca functia dvs de validare a campului se poate schimba in timpul prelucrarii formurilor.

#### O\_STATIC

Controleaza daca campul este fixat la dimensiunile initiale. Daca bitul este dezactivat, campul devine dinamic si isi va modifica dimensiunea pentru a incapa datele de intrare.

Optiunile unui camp nu pot fi schimbate in timp ce camp este selectat in momentul current. Oricum, optiunile pot fi schimbate in campuri afisate si care nu-s selectate in momentul curent

Optiunile sunt masti de biti si pot fi compuse cu sau logic in mod usual. Ati vazut cum se deactiveaza optiunea O\_AUOSKIP. Exemplul urmatoare clarifica modul de folosire al mai multor optiuni. Alte optiuni sunt explicate unde e cazul.

### Exemplu pentru optiuni ale campului

```
/* File Path: forms/form_options.c */
#include <form.h>

#define STARTX 15
#define STARTY 4
#define WIDTH 25

#define N_FIELDS 3

int main()
{
    FIELD *field[N_FIELDS];
    FORM *my_form;
    int ch, i;

    /* Initialize curses */
    initscr();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* Initialize the fields */
    for(i = 0; i < N_FIELDS - 1; ++i)
        field[i] = new_field(1, WIDTH, STARTY + i * 2, STARTX, 0, 0);
    field[N_FIELDS - 1] = NULL;

    /* Set field options */
    set_field_back(field[1], A_UNDERLINE);          /* Print a line for the
option */

    field_opts_off(field[0], O_ACTIVE); /* This field is a static label
*/
    field_opts_off(field[1], O_PUBLIC); /* This field is like a password
field*/
    field_opts_off(field[1], O_AUTOSKIP); /* To avoid entering the same
field */
}
```

```

/* after last character is
entered */

/* Create the form and post it */
my_form = new_form(field);
post_form(my_form);
refresh();

set_field_buffer(field[0], 0, "This is a static Field");
mvprintw(STARTY, STARTX - 10, "Field 1:");
mvprintw(STARTY + 2, STARTX - 10, "Field 2:");
refresh();

/* Loop through to get user requests */
while((ch = getch()) != KEY_F(1))
{
    switch(ch)
    {
        case KEY_DOWN:
            /* Go to next field */
            form_driver(my_form, REQ_NEXT_FIELD);
            /* Go to the end of the present buffer */
            /* Leaves nicely at the last character */
            form_driver(my_form, REQ_END_LINE);
            break;
        case KEY_UP:
            /* Go to previous field */
            form_driver(my_form, REQ_PREV_FIELD);
            form_driver(my_form, REQ_END_LINE);
            break;
        default:
            /* If this is a normal character, it gets */
            /* Printed */
            form_driver(my_form, ch);
            break;
    }
}

/* Un post form and free the memory */
unpost_form(my_form);
free_form(my_form);
free_field(field[0]);
free_field(field[1]);

endwin();
return 0;
}

```

Cu toate ca este nefolositor, acest exemplu arata cum pot fi utilizate optiunile. Folosite corespunzator, ele pot prezenta foarte eficient informatii intr-un form.. Cum al doilea camp nu este O\_PUBLIC, el nu arata caracterele pe care le tastati.

## Starea campurilor

Starea campurilor specifica daca respectivul camp a fost editat sau nu .Initial este FALSE si cand utilizatorul introduce ceva iar bufferul de date este modificat atunci devine TRUE. Prin

urmarea starea unui camp poate fi interogata pentru a afla daca a fost modificat sau nu. Urmatoarele functii ne pot asista in aceste operatii.

```
int set_field_status(FIELD *field, /*camp de modificat*/
                    int status); /*stre de setat*/

int field_status(FIELD *field); /*intoarce starea campului*/
```

E bine sa verificati starea campului doar dupa ce ati parasit campul, deoarece e posibil ca bufferul de date sa nu fie actualizat atata timp cat validarea inca are loc. Pentru a garanta ca este returnata starea corecta, apelati field\_status() fie cu (1) la rutina de validare a iesirii campului, (2) la iesirile utilizatorului de la initializarea sau terminare form-ului, sau (3) imediat dupa ce a fost procesata o cerere REQ\_VALIDATION de forms\_driver.

## Pointerul utilizator al campurilor

Fiecare structura de tip camp contine un pointer ce poate fi folosit de utilizator in scopuri diferite. Nu este atins de biblioteca formurilor si poate fi folosita in orice scop de utilizator. Functiile urmatoare seteaza si returneaza pointeri utilizator.

```
int set_field_userptr(FIELD *field,
                     char *userptr); /*pointer-ul utilizator pe care
                                     /*doriti sa-l asociati campului*/

char *field_userptr(FIELD *field); /*returneaza pointerul utilizator*/
                                     /*al campului */
```

## Campuri a caror marime depinde de variabile

Daca doriti un camp ce se modifica dinamic si are si latime variabila, atunci aceasta este componenta pe care vreti sa o utilizati la maxim. Va permite utilizatorului sa introduca mai multe date decat marimea initiala a campului si permite campului sa creasca. In functie de orientarea campului se va derula orizontal sau vertical pentru a incorpora noile date. Pentru a permite unui camp sa creasca dynamic, trebuie dezactivata optiunea O\_STATIC. Acest lucru poate fi facut cu field\_opts\_off(field\_pointer, O\_STATIC);

Dar in mod uzual nu este recomandat ca un camp sa creasca infinit. Puteti fixa o limita maxima pentru cresterea unui camp cu

```
int set_max_field(FIELD *field, /* campul in care operam*/
                  int max_growth); /*marimea maxima permisa*/
```

In cazul unui camp ce creste dinamic, putem afla informatii despre camp cu

```
int dynamic_field_info( FIELD *field, /* campul in care operam */
                       int *prows, /*nuamrul maxim de linii ce vor*/
                           /*fi completate*/
                       int *pcols, /*nuamrul maxim de coloane ce vor*/
```

```
/*fi completate*/
```

```
int *pmax) /*marimea maxima ce va fi */  
/*permisa pentru completare*/
```

Cu toate ca field\_info lucreaza ca de obicei, este indicat ca sa folosim aceasta functie pentru a afla atributele corespunzatoare pentru un camp a carui dimensiune creste dinamic.

Aduceti-va aminte de rutina de biblioteca new\_field; un camp nou creat cu inaltimea setata la unu va fi definit astfel incat sa fie un camp cu o singura linie. Daca este creat un camp cu inaltimea mai mare decat unu va fi definit in asa fel incat sa fie multiliniar.

Un camp uniliniar cu O\_STATIC dezactivat (camp cu crestere dinamica in dimensiune) va contine o singura linie fixa, dar numarul coloanelor poate creste in cazul in care utilizatorul introduce mai multe date decat poate primi campul initial. Numarul de coloane afisate va ramane fix iar prin datele aditionale vom putea derula orizontal.

Un camp multiliniar cu O\_STATIC dezactivat (camp cu crestere dinamica in dimensiune) va contine un numar fix de coloane, dar numarul de linii poate fi crescut in cazul in care utilizatorul introduce mai multe date decat poate primi campul initial. Numarul de linii afisate va ramane fix iar prin datele aditionale vom putea derula vertical.

Cele doua paragrafe de mai sus descriu in mare masura comportamentul unui camp a carui dimensiune creste dynamic. Modul in care se comporta alte parti ale bibliotecii form-urilor este descries mai jos:

1. Optiunea O\_AUTOSKIP va fi ignorata daca optiunea O\_STATIC este dezactivata si daca nu este specificata o marime maxima de crestere. In general, O\_AUTOSKIP genereaza automat o cerere REQ\_NEXT\_FIELD a form driverului in momentul in care utilizatorul tipareste in ultima portiune a campului. In cazul unui camp care creste si nu are specificata marimea maxima, nu exista ultima portiune a campului. Daca este specificata cresterea maxima, optiunea O\_AUTOSKIP va lucra in acelasi fel si in cazul in care campul a atins marimea maxima.
2. Alinierea campului va fi ignorata daca optiunea O\_STATIC este dezactivata. In general, set\_field\_just poate fi folosita doar in cazul JUSTIFY\_LEFT, JUSTIFY\_RIGHT, JUSTIFY\_CENTER pentru continutul unei linii a campului. Un camp uniliniar ce poate creste in dimensiune, prin definitie, creste si estederulabil orizontal si poate contine mai multe informatii decat pot fi alinate. Rezultatul de la field\_just va fi neschimbat.
3. Cererea supraincarcata a form driverului REQ\_NEW\_LINE va lucra in acelesi fel indiferent de optiunea campului O\_NL\_OVERLOAD chiar daca O\_STATIC este dezactivata si nu este precizata marimea maxima pentru camp. Uzual, daca este activata O\_NL\_OVERLOAD, atunci REQ\_NEW\_LINE genereaza implicit REQ\_NEXT\_FIELD daca este apelat din ultima linie a campului. Daca un camp poate creste fara limita, atunci nu exista ultima linie, deci REQ\_NEW\_LINE nu va genera implicit REQ\_NEXT\_FIELD. Daca este specificata o limita maxima de crestere si optiunea form-ului O\_NL\_OVERLOAD este activata, atunci REQ\_NEW\_LINE va genera doar implicit REQ\_NEXT\_FIELD in cazul in care campul a crescut pana la limita maxima iar utilizatorul a ajuns la ultima linie.

4. Apelul functiei de biblioteca `call_dup_field` va functiona ca de obicei; va duplica campul, inclusive marimea bufferului curent si continutul campului este si el duplicate. Orice crestere maxima specificata va fi si ea duplicate
5. Apelul bibliotecii functiei de biblioteca `link_field` va functiona ca de obicei; va duplica toate attributele campului si se va face o legatura intre bufferele partajate ale campului. In cazul in care optiunea campului `O_STATIC` este schimbata anterior de buffere ce partajeaza un camp, modul in care sistemul reactioneaza la o incercare de a introduce mai multe date decat are bufferul in prezent va depinde de setarea optiunii in campul actual.
6. Apelul functiei de biblioteca `field_info` va functiona ca de obicei; variabila `nrow` va contine valoarea apelului initial al `new_field`. Utilizatorul ar trebui sa foloseasca `dynamic_field_info`, descries mai sus, pentru a interoga marimea curenta a bufferului.

Unele dintre punctele de mai sus au sens doar dupa explicatii in privinta lui form driver. Ne vom ocupa de acest lucru in sectiunile urmatoare.

## Form-uri de tip ferestre

Conceptul de form de tip fereastră este in mare parte similar cu ferestre meniu. Fiecare form este asociat cu o fereastră principala si cu subfereastră. Fereastră principala a form-ului afiseaza orice titlu sau margine asociata sau orice doreste utilizatorul. Subfereastră contine toate campurile si le afiseaza in concordanta cu pozitia lor. Acest lucru ne ofera flexibilitate in manipularea cu usurinta a afisarii formurilor dichisite. Din moment ce este in mare parte similar cu ferestre meniu, va ofer un exemplu cu multe explicatii. Fnctiile sunt similare si lucreaza la fel.

## Exemplu pentru form-uri de tip ferestre

```
/* File Path: forms/form_win.c */
#include <form.h>

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color);

int main()
{
    FIELD *field[3];
    FORM *my_form;
    WINDOW *my_form_win;
    int ch, rows, cols;

    /* Initialize curses */
    initscr();
    start_color();
    cbreak();
    noecho();
    keypad(stdscr, TRUE);

    /* Initialize few color pairs */
    init_pair(1, COLOR_RED, COLOR_BLACK);
```

```

/* Initialize the fields */
field[0] = new_field(1, 10, 6, 1, 0, 0);
field[1] = new_field(1, 10, 8, 1, 0, 0);
field[2] = NULL;

/* Set field options */
set_field_back(field[0], A_UNDERLINE);
field_opts_off(field[0], O_AUTOSKIP); /* Don't go to next field when
this */

/* Field is filled up
*/
set_field_back(field[1], A_UNDERLINE);
field_opts_off(field[1], O_AUTOSKIP);

/* Create the form and post it */
my_form = new_form(field);

/* Calculate the area required for the form */
scale_form(my_form, &rows, &cols);

/* Create the window to be associated with the form */
my_form_win = newwin(rows + 4, cols + 4, 4, 4);
keypad(my_form_win, TRUE);

/* Set main window and sub window */
set_form_win(my_form, my_form_win);
set_form_sub(my_form, derwin(my_form_win, rows, cols, 2, 2));

/* Print a border around the main window and print a title */
box(my_form_win, 0, 0);
print_in_middle(my_form_win, 1, 0, cols + 4, "My Form",
COLOR_PAIR(1));

post_form(my_form);
wrefresh(my_form_win);

mvprintw(LINES - 2, 0, "Use UP, DOWN arrow keys to switch between
fields");
refresh();

/* Loop through to get user requests */
while((ch = wgetch(my_form_win)) != KEY_F(1))
{
    switch(ch)
    {
        case KEY_DOWN:
            /* Go to next field */
            form_driver(my_form, REQ_NEXT_FIELD);
            /* Go to the end of the present buffer */
            /* Leaves nicely at the last character */
            form_driver(my_form, REQ_END_LINE);
            break;
        case KEY_UP:
            /* Go to previous field */
            form_driver(my_form, REQ_PREV_FIELD);
            form_driver(my_form, REQ_END_LINE);
            break;
        default:

```

```

        /* If this is a normal character, it gets */
        /* Printed                                     */
        form_driver(my_form, ch);
        break;
    }
}

/* Un post form and free the memory */
unpost_form(my_form);
free_form(my_form);
free_field(field[0]);
free_field(field[1]);

endwin();
return 0;
}

void print_in_middle(WINDOW *win, int starty, int startx, int width, char
*string, chtype color)
{
    int length, x, y;
    float temp;

    if(win == NULL)
        win = stdscr;
    getyx(win, y, x);
    if(startx != 0)
        x = startx;
    if(starty != 0)
        y = starty;
    if(width == 0)
        width = 80;

    length = strlen(string);
    temp = (width - length) / 2;
    x = startx + (int)temp;
    wattron(win, color);
    mvwprintw(win, y, x, "%s", string);
    wattroff(win, color);
    refresh();
}

```

## Validarea campurilor

Implicit, un camp va accepta orice date introduse de utilizator. Este posibil sa atasam validare unui camp. Apoi orice incercare a utilizatorului de a parasi campul va esua, atata timp cat campul contine date care nu corespund tipului de validare. Unele tipuri de validare au si o verificare a caracterelor in fiecare moment in care un caracter este introdus in camp.

Validarea poate fi atasata unui camp cu functia urmatoare.

```

int set_field_type(FIELD *field, /*camp de modificat*/
                  FIELDTYPE *ftype, /*tip ce trebuie asociat*/
                  ...); /*argumente aditionale*/

```

Odata fixat, tipul validarii pentru un camp poate fi interogata cu

```
FIELDTYPE *field_type(FIELD *field);          /*camp ce trebuie interogat*/
```

Form driverul valideaza datele intr-un camp doar cand datele sunt introduse de un utilizator „final”. Validarea nu are loc atunci cand

- aplicatia schimba valoarea campului apeland calling set\_field\_buffer.
- valorile ce sunt legate de un camp sunt modificate indirect –schimband valoarea campului

Urmatoarele sunt tipuri de validare predefinite.Puteti de asemenea specifica validare particularizata,dar este cam impovaratoare.

## **TYPE\_ALPHA**

Acest tip de camp accepta date alfabetice; fara goluri,cifre,caractere speciale(acest lucru este verificat in timpul introducerii caracterelor).Este fixat cu :

```
int set_field_type(FIELD *field,          /*camp ce trebuie modificat*/
                  TYPE_ALPHA,           /*tipul asociat*/
                  int width);          /*largimea maxima a campului*/
```

Argumentul width seteaza largimea minima a datelor. Utilizatorul trebuie sa introduca cel putin numarul de caractere specificat de acest argument inainte de a parasi campul.In mod caracteristic veti dori sa fixati aceasta valoare la largimea campului; daca este mai mare decat largimea campului evaluarea validarii va esua mereu .Largimea minima zero face optionala completarea campului.

## **TYPE\_ALNUM**

Acest tip de camp accepta date alfabetice si cifre; fara goluri,cifre,caractere speciale(acest lucru este verificat in timpul introducerii caracterelor) Este fixat cu :

```
int set_field_type(FIELD *field,          /*camp ce trebuie modificat*/
                  TYPE_ALNUM,           /* tipul asociat */
                  int width);          /*largimea maxima a campului*/
```

Argumentul width seteaza largimea minima a datelor.In cazul TYPE\_ALPHA,veti dori sa-l fixati la largimea minima; daca este mai mare decat largimea campului evaluarea validarii va esua mereu. Largimea minima zero face optionala completarea campului.

## **TYPE\_ENUM**

Permite restrictionarea valoriiilor campului astfel incat sa fie intr-o multime specificata de string-uri (de exemplu,codurile postale din doua caractere din U.S.).Este fixat cu:

```
int set_field_type(FIELD *field,          /*camp de modificat*/
                  TYPE_ENUM,           /*tip asociat*/
                  char **valuelist;     /*lista de valori posibile*/
```

```

int checkcase;           /*case-sensitive*/
int checkunique); /*trebuie specificat in mod unic*/

```

Parametrul lista de valori trebuie sa poarte la o lista de string-uri terminata cu NULL. Argumentul checkcase, daca este setat true, face comparatia cu toate string-urile care sunt in lista.

Cand utilizatorul iese dintr-un camp de tipul TYPE\_ENUM, procedura de validare incearca sa completeze data din buffer cu una valida. Daca a fost introdus un string complet din multimea din care avem de ales, atunci el este valid. Dar este de asemenea posibil sa introducem un prefix al unui string valid si sa fie completat in locul dumneavoastra. Implicit, daca introduci un asemenea prefix si corespunde cu mai multe valori din lista, prefixul va fi completat cu prima valoare care se potriveste. Dar daca argumentul checkunique este true, atunci trebuie potrivirea prefixului sa fie identica pentru a fi valida. Cererile de intrare REQ\_NEXT\_CHOICE si REQ\_PREV\_CHOICE pot fi folositoare in cazul acestor campuri.

## TYPE\_INTEGER

Acest camp accepta un intreg. Se seteaza dupa cum urmeaza:

```

int set_field_type(FIELD *field,           /*camp de modificat*/
                  TYPE_INTEGER,           /*tip asociat*/
                  int padding, /*#locuri care trebuiesc facute zero*/
                  int vmin, int vmax); /*interval valid*/

```

Caracterele valide constau dintr-un minus in fata si cifre. Verificarea intervalului este facuta la iesire. Daca maximul intervalului este mai mic sau egal cu minimul, atunci intervalul este ignorat. In cazul in care valoarea trece de testul intervalului, atunci se pun in fata sa atatea zerouri cate au fost specificate in argumentul "padding".

Un buffer al unei valori de tip TYPE\_INTEGER poate fi interpretat cu usurinta de functia atoi(3) a bibliotecii C..

## TYPE\_NUMERIC

Acest camp accepta un numar zecimal. Este setat in felul urmatoar:

```

int set_field_type(FIELD *field,           /*camp de modificat*/
                  TYPE_NUMERIC,           /*tip asociat*/
                  int padding, /* #nr de zerouri de dupa virgula*/
                  int vmin, int vmax); /*interval valid*/

```

Caracterele valide sunt un minus optional si cifre, cu posibilitatea includerii unei virgule zecimale. Verificarea intervalului are loc la iesire. Daca maximul intervalului este mai mic sau egal cu minimul, atunci intervalul este ignorat. Daca valoarea trece de verificarea intervalului, ii sunt adaugate in coada numarul de zerouri astfel incat sa satisfaca argumentul padding.

Buffer-ul unei valori TYPE\_NUMERIC poate fi interpretat convenabil de functia atof(3) a bibliotecii C.

## TYPE\_REGEX

Acest camp accepta date care corespund expresiilor regulate. Este setat dupa cum urmeaza:

```
int set_field_type(FIELD *field,          /*camp de alterat*/
                  TYPE_REGEX,           /*tip asociat*/
                  char *regex);         /*expresie cu care sa se potriveasca*/
```

Sintaxa pentru expresii regulate este aceea a regcomp(3). Verificarea pentru potrivirea expresiilor regulate are loc la iesire.

## Formularul principal: Baza formularelor sistemului

Ca si in cazul meniului sistem, form\_driver() joaca un rol important in sistemul form-urilor. Toate tipurile de cereri catre sistemul form-urilor ar trebui sa treaca prin form\_driver().

```
int form_driver(FORM *form,          /*form-ul cu care lucram */
               int request)         /*cod pentru cerere a form-ului */
```

Ca si in exemplul de mai sus, trebuie sa va aflati intr-o bucla cautand date de intrare de la utilizator si apoi sa decidem daca este o data pentru camp sau o cerere a form-ului. Apoi cererile form-ului sunt trimise catre form\_driver() pentru a-si face treaba. Cererile sunt impartite, intr-o maniera grosolana, in urmatoarele categorii. Mai jos sunt prezentate cereri diferite si modul de utilizare.

## Cereri de navigare paginata

Aceste cereri cauzeaza deplasari cu nivelori de pagina in cadrul form-ului, cauzand afisarea unui nou ecran al form-ului. Un form poate fi creat cu pagini multiple. Daca aveti un form mare cu multe campuri si sectiuni logice, atunci puteti impartii form-ul in pagini. Functia set\_new\_page() este utilizata pentru a seta o pagina noua la campul specificat.

```
int set_new_page(FIELD *field, /*campul la care sa se fixeze/scoata o
intrerupere de pagina */
                bool new_page_flag); /*ar trebui sa fie TRUE pentru o
intrerupere*/
```

Cererile urmatoare permit deplasare pe alta pagina:

- *REQ\_NEXT\_PAGE*  
Muta la urmatoare pagina a form-ului
- *REQ\_PREV\_PAGE*  
Muta la pagina anterioara a form-ului
- *REQ\_FIRST\_PAGE*  
Muta la prima pagina a form-ului
- *REQ\_LAST\_PAGE*  
Muta la ultima pagina a form-ului

Aceste cereri trateaza lista ciclic, adica, REQ\_NEXT\_PAGE de la ultima pagina se muta la prima, si REQ\_PREV\_PAGE de la prima pagina se muta la ultima.

## Cereri de navigare intre campuri

Aceste cereri trateaza navigarea in cazul campurilor aceleiasi pagini

- *REQ\_NEXT\_FIELD*  
Deplaseaza la campul urmator.
- *REQ\_PREV\_FIELD*  
Deplaseaza la campul anterior
- *REQ\_FIRST\_FIELD*  
Deplaseaza la primul camp.
- *REQ\_LAST\_FIELD*  
Deplaseaza la ultimul camp.
- *REQ\_SNEXT\_FIELD*  
Deplaseaza la urmatorul camp sortat.
- *REQ\_SPREV\_FIELD*  
Deplaseaza la campul anterior sortat.
- *REQ\_SFIRST\_FIELD*  
Deplaseaza la primul camp sortat.
- *REQ\_SLAST\_FIELD*  
Deplaseaza la ultimul camp sortat.
- *REQ\_LEFT\_FIELD*  
Deplaseaza la stanga campului
- *REQ\_RIGHT\_FIELD*  
Deplaseaza la dreapta campului
- *REQ\_UP\_FIELD*  
Deplaseaza deasupra campului
- *REQ\_DOWN\_FIELD*  
Deplaseaza dedesuptul campului.

Aceste cereri trateaza ciclic lista de campuri de pe o pagina, adica, *REQ\_NEXT\_FIELD* de la ultimul camp merge la primul, iar *REQ\_PREV\_FIELD* de la primul camp merge la ultimul. Ordinea pentru aceste cereri (si pentru cererile *REQ\_FIRST\_FIELD* si *REQ\_LAST\_FIELD*) este pur si simplu ordinea pointerilor campurilor in vectorul de form-uri (asa cum sunt setati de *new\_form()* sau *set\_form\_fields()*).

Este posibila si traversarea campurilor in ordinea pozitionarii pe ecran., asa ca secventa merge dreapta-stanga si de sus in jos. Pentru a putea face acest lucru, utilizati al doilea grup din cele patru cereri sortate in functie de miscare. In cele din urma, este posibila deplasare intre campuri folosind instructiunile vizuale sus, jos, dreapta, si stanga. Pentru a putea face acest lucru, utilizati al treilea grup din cele patru cereri. Tineti cont de faptul ca , pentru a folosi aceste cereri, pozitionare form-ului este caoltul stanga sus. De exemplu, sa presupunem ca aveti un camp multiliniar B, si doua campuri monoliniare A si C pe aceeasi linie cu B, cu A la stanga lui B si C la dreapta lui B. O cerere *REQ\_MOVE\_RIGHT* de la A va merge la B doar daca A, B, si C partajeaza cu totii aceeasi prima linie; altfel va sari peste B la C.

## Cereri de navigare in interiorul campului

Aceste cereri coordoneaza miscarea cursorului de editare in interiorul campului curent selectat.

- *REQ\_NEXT\_CHAR*  
Depleaza la urmatorul caracter.

- *REQ\_PREV\_CHAR*  
Depleaza la caracterul anterior.
- *REQ\_NEXT\_LINE*  
Depleaza la urmatoarea linie.
- *REQ\_PREV\_LINE*  
Depleaza la linia anterioara.
- *REQ\_NEXT\_WORD*  
Depleaza la cuvantul urmator.
- *REQ\_PREV\_WORD*  
Depleaza la cuvantul anterior.
- *REQ\_BEG\_FIELD*  
Deplasare la inceputul campului.
- *REQ\_END\_FIELD*  
Deplasare la inceputul campului.
- *REQ\_BEG\_LINE*  
Deplasare la inceputul liniei.
- *REQ\_END\_LINE*  
Deplasare la sfarsitul liniei.
- *REQ\_LEFT\_CHAR*  
Deplasare la stanga in interiorul campului.
- *REQ\_RIGHT\_CHAR*  
Deplasare la dreapta in interiorul campului.
- *REQ\_UP\_CHAR*  
Deplasare in sus in interiorul campului.
- *REQ\_DOWN\_CHAR*  
Deplasare in jos in interiorul campului.

Fiecare cuvant este separat de caracterele antrerioare si urmatoare de spatiu. Comenzile pentru deplasare la inceptul si sfarsitul liniei sau al campului cauta primul sau ultimul caracter din intervalul lor care nu este zona tampon (de ex. nu este spatiu).

## **Cereri de derulare**

Campurile care sunt dinamice si au si-au modificat dimensiune si cele care au fost create explicit cu linii in afara ecranului sunt derulabile. Campurile monolineare sunt derulabile pe orizontala; cele cu mai multe linii sunt derulabile pe verticala. Cele mai dese derulari sunt declansate de editare si deplasarea in interiorul campului (biblioteca deruleaza campul pentru a mentine cursorul vizibil). Este posibila o solicitare explicita a derularii folosind urmatoarele cereri:

### *REQ\_SCR\_FLINE*

Deruleaza vertical o linie inainte

- *REQ\_SCR\_BLINE*

Deruleaza vertical o linie inapoi

- *REQ\_SCR\_FPAGE*  
Deruleaza vertical o pagina inainte
- *REQ\_SCR\_BPAGE*  
Deruleaza vertical o pagina inapoi
- *REQ\_SCR\_FHPAGE*  
Deruleaza vertical o jumatate de pagina inainte
- *REQ\_SCR\_BHPAGE*  
Deruleaza vertical o jumatate de pagina inapoi
- *REQ\_SCR\_FCHAR*  
Deruleaza orizontal un caracter inainte.
- *REQ\_SCR\_BCHAR*  
Deruleaza orizontal un caracter inapoi.
- *REQ\_SCR\_HFLINE*  
Deruleaza orizontal o latime de pagina inainte.
- *REQ\_SCR\_HBLINE*  
Deruleaza orizontal o latime de pagina inapoi.
- *REQ\_SCR\_HFHALF*  
Deruleaza orizontal o jumatate de latime de pagina inainte
- *REQ\_SCR\_HBHALF*  
Deruleaza orizontal o jumatate de latime de pagina inapoi

In scopul derularii, o pagina sau un camp este inaltimea partii sale vizibile.

## Cereri de editare

Cand trimiteti driverul form-urilor un caracter ASCII , este tratat ca o cerere de adaugare a caracterului in buffer-ul de date al campului. Daca este o insertie sau o inlocuire depinde de modul de editare al campului( implicit este insertie).

Urmatoarele cereri suporta editarea campului si schimbarea modului de editare:

- *REQ\_INS\_MODE*  
Seteaza modul de insertie.
- *REQ\_OVL\_MODE*  
Set modul de suprapunere.
- *REQ\_NEW\_LINE*  
Cerere pentru o linie noua(explicatii mai jos).
- *REQ\_INS\_CHAR*  
Insereaza spatiu in locatia caraterului.
- *REQ\_INS\_LINE*  
Insereaza linie goala in locatia caraterului.
- *REQ\_DEL\_CHAR*  
Sterge caracterul pe care e pozitionat cursorul.
- *REQ\_DEL\_PREV*  
Sterge caracterul anterior fata de pozitia cursorului.
- *REQ\_DEL\_LINE*  
Sterge linia pe care e pozitionat cursorul.
- *REQ\_DEL\_WORD*  
Sterge cuvantul pe care e pozitionat cursorul.
- *REQ\_CLR\_EOL*

Sterge capatul unei linii.

- *REQ\_CLR\_EOF*

Sterge capatul unui camp.

- *REQ\_CLEAR\_FIELD*

Sterge un camp intreg.

Comportamentul cererilor *REQ\_NEW\_LINE* si *REQ\_DEL\_PREV* este complicat si partial controla de o pereche de optiuni ale form-urilor. Cazurile speciale sunt declansate cand cursorul este la inceputul unui camp, sau la ultima linie a campului.

Prima data luam in considerare *REQ\_NEW\_LINE*:

Comportamentul normal al *REQ\_NEW\_LINE* in modul de insertie este de a intrerupe linia curenta la pozitia cursorului de editare, inserand portiunea liniei curente dupa cursor ca o linie noua in continuarea liniei curente si deplasand cursorul la inceputul liniei noi( e ca si cum ai insera o linie noua in buffer-ul campului).

Comportamentul normal al lui *REQ\_NEW\_LINE* in modul de suprapunere este de a sterge linia curenta din pozitia in care se afla cursorul pana la capatul liniei. Cursorul este deplasat la inceputul liniei urmatoare.

In orice caz, daca *REQ\_NEW\_LINE* este la inceputul campului , sau daca este pe ultima linie a campului, face o *REQ\_NEXT\_FIELD*. In cazul in care optiunea *O\_NL\_OVERLOAD* este dezactivata, actiunea speciala este dezactivata.

Acum sa vedem *REQ\_DEL\_PREV*:

Comportamentul normal al *REQ\_DEL\_PREV* este de a sterge caracterul anterior. Daca modul de insertie este activ, si cursorul este la inceputul liniei, iar textul de pe acea linie incapa pe linia precedenta, se pozitioneaza in schimb la sfarsitul continutului liniei curente si sterge portiunea de text pana la linia precedenta . (similar cu a sterge o linie noua din buffer-ul campului)you may think of this as deleting a newline from the field buffer).

In orice caz, *REQ\_DEL\_PREV* la inceputul campului este tratat ca o *REQ\_PREV\_FIELD*.

Daca optiunea *O\_BS\_OVERLOAD* nu este activa, aceasta actiune speciala este dezactivata iar driver-ul form-urilor intoarce doar *E\_REQUEST\_DENIED*.

## Cereri de ordonare

Daca tipul campului dumnevoastra este ordonat, si are functii asociate pentru a afla valorile urmatoare si anterioare tipului pentru o valoare data, exista cereri care pot returna acea valoare intr-un buffer al campului:

- *REQ\_NEXT\_CHOICE*

Plaseaza valoarea succesoarea a valorii curente in buffer.

- *REQ\_PREV\_CHOICE*

Plaseaza valoarea anterioara a valorii curente in buffer.

Dintre tipurile de campuri built-in , doar TYPE\_ENUM are functii built-in pentru sucesor si predecesor. Cand iti definesti un tip de camp(Custom Validation Types), Poti asocia functii proprii de ordonare.

## Comenzi ale aplicatiei

Cererile form-urilor sunt reprezentate ca intregi cu valoare mai mare decat vlorile din curses cu valoare mai mare decat KEY\_MAX si mai mica sau egal decat constanta MAX\_COMMAND.Valoarile din acest interval sunt ignorate de form\_driver(). Deci poate fi folosita in orice scop de aplicatie.Poate fi tratata ca actiune specifica a aplicatiei si se executa actiunea corespunzatoare.

## Bibliotecile Tools si Widget

Acum ca ati vazut capacitatile ncurses si ale bibliotecilor inrudite, va suflecati manecile si va inhamati la un proiect care manipuleaza din greu ecranul.Dar asteptati. Poate fi destul de dificil sa scrieti si sa mentineti widget-uri GUI in ncurses pur chiar si utilizand bibliotecile aditionale.Exista instrumente gata de a fi utilizate si biblioteci pentru widget-uri care pot fi folosite in loc de a scrie propriile widget-uri. Puteti folosi cateva, sa va inspirati din cod, sau chiar sa le extindeti.

## CDK (Curses Development Kit)

In cuvintele autorului

*CDK vine de la 'Curses Development Kit' si contine la momentul actual 21 widget-uri ca faciliteaza dezvoltarea cu viteza a programelor full screen ce folosesc curses.*

Acest kit ofera cateva widget-uri folositoare, care pot fi utilizate in mod direct in programele voastre. Sunt destul de bine scrise iar documentatia este foarte buna.Exemplele din directorul

de exemple pot fi, pentru incepatori, un loc bun de a incepe.CDK poate fi downloaded <http://www.vexus.ca/release/cdk.tar.gz> . Urmati instructiunile din fisierul README pentru instalare.

## Lista de Widget-uri

In continuare avem o lista de widget-uri oferite impreuna cu cdk si descrierea lor.

| Tip Widget          | Scurta descriere                                                                                                                                                               |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Alphalist           | Permite unui utilizator selectia dintr-o lista cu posibilitatea de a anulaselectia tiparind cateva litere din cuvantul ce trebuie selectat                                     |
| Buttonbox           | This creates a multiple button widget.                                                                                                                                         |
| Calendar            | Creates a little simple calendar widget.                                                                                                                                       |
| Dialog              | Prompts the user with a message, and the user can pick an answer from the buttons provided.                                                                                    |
| Entry               | Allows the user to enter various types of information.                                                                                                                         |
| File Selector       | A file selector built from Cdk base widgets. This example shows how to create more complicated widgets using the Cdk widget library.                                           |
| Graph               | Draws a graph.                                                                                                                                                                 |
| Histogram           | Draws a histogram.                                                                                                                                                             |
| Item List           | Creates a pop up field which allows the user to select one of several choices in a small field. Very useful for things like days of the week or month names.                   |
| Label               | Displays messages in a pop up box, or the label can be considered part of the screen.                                                                                          |
| Marquee             | Displays a message in a scrolling marquee.                                                                                                                                     |
| Matrix              | Creates a complex matrix with lots of options.                                                                                                                                 |
| Menu                | Creates a pull-down menu interface.                                                                                                                                            |
| Multiple Line Entry | A multiple line entry field. Very useful for long fields. (like a description field)                                                                                           |
| Radio List          | Creates a radio button list.                                                                                                                                                   |
| Scale               | Creates a numeric scale. Used for allowing a user to pick a numeric value and restrict them to a range of values.                                                              |
| Scrolling List      | Creates a scrolling list/menu list.                                                                                                                                            |
| Scrolling Window    | Creates a scrolling log file viewer. Can add information into the window while its running. A good widget for displaying the progress of something. (akin to a console window) |
| Selection List      | Creates a multiple option selection list.                                                                                                                                      |
| Slider              | Akin to the scale widget, this widget provides a visual slide bar to represent the numeric value.                                                                              |
| Template            | Creates a entry field with character sensitive positions. Used for pre-formatted fields like dates and phone numbers.                                                          |
| Viewer              | This is a file/information viewer. Very useful when you need to display loads of information.                                                                                  |

## Cateva trasaturi atractive

In afara ca ne usuram viata cu widget-uri rapide si folositoare, cdk rezolva problema frustranta a printarii unor siruri de caractere in culori multiple, aliniind sirurile de caractere intr-un mod elegant. Etichete formatare special pot fi aplicate stringurilor care sunt transmise functiilor CDK. De exemplu daca sirul

```
"</B/1>This line should have a yellow foreground and a blue background.<!1>"
```

dat ca parametru functiei newCDKLabel(), afiseaza linia cu prim-planul galben si fundalul albastru. Exista si alte etichete disponibile pentru alinierea sirurilor de caractere la stanga si la dreapta, continand caractere de desenare speciale etc. Consultati pagina de help `cdk_display(3X)` pentru detalii. Pagina de help explica utilizarea lor cu exemple dragute.

## Concluzii

Ca urmare, CDK este un pachet de widget-uri bine scris, care folosit corect poate construi un mediu de lucru puternic pentru dezvoltarea de GUI complexe.

## Dialogul

Foarte demult, in septembrie 1994, cand putini oameni stiau linux, Jeff Tranter a scris un [articol](#) despre dialoguri in Linux Journal. El incepe articolul cu aceste cuvinte..

*Linux este bazat pe sistemul de operare Unix, dar contine de asemenea un numar de caracteristici kernel unice si folositoare si programe de aplicatii care de multe ori depasesc lucrurile disponibile in Unix. O valoare putin cunoscuta este "dialog", o utilitate pentru crearea de boxe de dialog profesionale plecand de la scripturi shell. Acest articol prezinta un tutorial introductiv despre utilitatea dialogului si arata exemple despre cum si unde pot fi folosite.*

Dupa cum el explica, dialogul este un adevarat jiuvaer in crearea cu usurinta de boxe de dialog cu infatisare profesionala. Acesta creaza o varietate de boxe de dialog, meniuri, check liste etc. De obicei este instalat implicit. Daca nu, o puteti gasi la [ibiblio linux archive](#).

Articolul mentionat mai sus reda o imagine buna asupra utilizarilor si capacitatile sale. Pagina man are mai multe detalii. Poate fi folosit in situatii variate. Un exemplu bun este construirea nucleului linux in mod text. Nucleul Linux foloseste o versiune modificata de dialoguri ajustata dupa nevoile sale.

Dialogul a fost initial proiectat pentru a fi folosit cu scripturi shell. Daca vreti sa ii folositi functionalitatea intr-un program C, puteti folosi libdialog. Documentatia referitoare la aceasta este rara. Referinta definitiva este fisierul header dialog.h care vine o data cu biblioteca. Ati avea probabil nevoie sa munciti mult aici si apoi sa obtineti iesirea dorita. Sursa este usor de customizat. Am folosit-o intr-un numar de ocazii prin modificarea codului.

## Modulele Perl Curses CURSES::FORM si CURSES::WIDGETS

Modulul Curses in perl, Curses::Form si Curses::Widgets da acces la curses prin perl. Daca aveti curses si este instalat basic perl, poti obtine aceste module de la [CPAN All Modules page](#). In cele trei module arhivate din categoria Curses. O data instalate poti folosi aceste module din scripurile perl la fel ca orice alte module. Pentru mai multe informatii despre modulele perl vezi pagina man perlmod. Modulele de mai sus vin cu o documentatie buna si au cateva scripturi demo pentru a testa functionalitatea. Desi widget-urile furnizate sunt foarte rudimentare, aceste module furnizeaza un acces bun la bibliotecile curses din perl.

Pentru mai multe informatii vezi paginile man Curses(3), Curses::Form(3) si Curses::Widgets(3). Aceste pagini sunt instalate numai cand modulele de mai sus sunt obtinute si instalate.

## Distractiv!!!

Aceasta sectiune contine cateva programe scrise de mine doar de distractie. Ele nu reprezinta un mod de programare mai practic sau cel mai bun mod de folosire a ncurses. Ele apar aici doar pentru a permite incepatorilor sa isi faca o idee si pentru a adauga mai multe program acestei sectiuni. Daca ati scris cateva program dragute si simple in curses si vreti sa le includeti aici , contactati-ma [me](#).

### *Jocul vietii*

Jocul vietii este o minune a matematicii. In cuvintele lui [Paul Callahan](#)

Jocul vietii (sau simplu Life) nu este un joc in sens conventional. Nu exista jucatori, si nici castigator sau infrant. O data ce piesele sunt puse in pozitia de start, regulile determina ce se va intampla mai tarziu. Totusi Life este pln de surprize! In cele mai multe cazuri, este imposibil sa te uiti la o pozitie de start(sau pattern) si sa iti dai seama ce se va intampla in viitor. Singurul mod de a-ti da seama este sa urmezi regulile jocului.

Acest program incepe cu un simplu pattern U inversat si arata cat de minunat functioneaza viata. Exista loc pentru multe imbunatatiri in program. Poti lasa utilizatorul sa isi introduca patternul dorit sau chiar sa iei intrarea dintr-un fisier. Poti schimba regulile si sa joci in mai multe variante. Cauta pe [google](#) informatii interesante despre jocul vietii.

Cale fisier: JustForFun/life.c

### *Patratul magic*

Magic Square, o alta minune a matematicii, este foarte usor de inteles dar foarte greu de realizat. Intr-un patrat magic suma numerelor din fiecare linie, fiecare coloana este egal. Chiar si suma diagonalelor este egala. Exista multe variatii care au proprietati speciale.

Acest program creaza un simplu patrat magic de ordine impara.

*Cale fisier: JustForFun/magic.c*

### *Turnurile din Hanoi*

Rezolvarea problemei turnurile din Hanoi. Baza jocului reprezinta mutarea discurilor de pe primul tarus pe ultimul, folosind un tarus de mijloc pentru stationare temporara. Regula este sa nu plasezi un disc mai mare peste un disc mai mic la nici un moment.

*Cale fisier:: JustForFun/hanoi.c*

### *Puzzelul Queens*

Obiectivul pentru faimosul puzzel N-Queen este sa pui N regine pe o tabla de sah NxN fara a se ataca una pe cealalta.

Acest program o rezolva printr-o simpla tehnica de backtracking

*Cale fisier: JustForFun/queens.c*

### *Shuffle*

Un joc distractiv daca ai timp liber.

*Cale fisier: JustForFun/shuffle.c*

### *Tutorial de tastat*

Un simplu tutorial de tastat, l-am creat mai mult din nevoie decat pentru confortul folosirii. Daca stiii cum sa iti pui degetele corect pe tastatura, dar nu ai practica, acesta poate sa fie folositor.

*Cale fisier: JustForFun/tt.c*

