Unelte de programare

Responsabil: Laura Vasilescu [mailto:laura.vasilescu@cti.pub.ro]

Obiective

În urma parcurgerii acestui laborator studentul va fi capabil să:

- scrie și compileze programe simple în limbajul C
- utilizeze programul Make pentru a compila automat programele scrise
- utilizeze funcțiile din limbaj de citire și scriere a datelor

Introducere

C este un limbaj de programare structurată menit să simplifice scrierea programelor apropiate de masină. A fost creat de către Dennis Ritchie în perioada 1968-1973 și a fost dezvoltat în strânsă legatură cu sistemul de operare Unix, care a fost rescris în întregime în C. Utilizarea limbajului s-a extins cu trecerea timpului de la sisteme de operare și aplicaţii de sistem la aplicaţii generale.

Deşi în prezent, pentru dezvoltarea aplicaţiilor complexe au fost create limbaje de nivel mai înalt (`Java, C#, Python`), C este în continuare foarte folosit la scrierea sistemelor de operare şi a aplicațiilor de performanţă mare sau dimensiune mică (în lumea dispozitivelor embedded). Nucleele sistemelor Windows şi Linux sunt scrise în C.

Unelte folosite

Compilatorul GCC

În cadrul laboratorului și pentru testarea temelor de casă se va folosi compilatorul GCC. GCC este unul dintre primele pachete software dezvoltate în cadrul Proiectului GNU (`GNU's Not Unix`) de către Free Software Foundation. Deși GCC se traducea iniţial prin `GNU C Compiler`, acesta a devenit între timp un compilator multifrontend, multi-backend, având suport pentru o serie largă de limbaje, ca C, C++, Objective-C, Ada, Java, etc, astfel că denumirea curentă a devenit `GNU Compiler Collection`. În cadrul cursului de Programare [http://ocw.cs.pub.ro/programare-ca] ne vom referi totuşi numai la partea de C din suita de compilatoare.

Compilatorul GCC rulează pe o gamă largă de echipamente hardware (procesoare din familia: `i386, alpha, vax, m68k, sparc, HPPA, arm, MIPS, PowerPC,` etc.) și de sisteme de operare (`GNU/Linux, DOS, Windows 9x/NT/2000, Solaris, Tru64, VMS, Ultrix, Aix`), fiind la ora actuală cel mai folosit compilator.

Compilatorul GCC se apelează din linia de comandă, folosind diferite opțiuni, în funcție de rezultatul care se dorește (specificarea de căi suplimentare de căutare a bibliotecilor/fișierelor antet, link-area unor biblioteci specifice, opțiuni de optimizare, controlul stagiilor de compilare, al avertismentelor, etc.).

Utilizare GCC

Vom folosi pentru exemplificare un program simplu care tipărește la ieșirea standard un șir de caractere.

hello.c

```
#include <stdio.h>

int main(void) {
  printf("Hello from your first program!\n");

  return 0;
}
```

Pentru compilarea programului se va lansa comanda (în linia de comandă):

```
gcc hello.c
```

presupunând că fișierul sursă se numește `hello.c`.

Pe un sistem de operare `Linux`, compilarea default va genera un executabil cu numele `a.out`. Pentru rularea acestuia, trebuie executată comanda:

```
./a.out
```

Pentru un control mai fin al comportării compilatorului, sunt prezentate în tabelul următor cele mai folosite opţiuni (pentru lista completă studiaţi pagina de manual pentru GCC - `man gcc`):

Opțiune	Efect
-o nume_fișier	Numele fișierului de ieşire va fi nume_fişier. În cazul în care această opțiune nu este setată, se va folosi numele implicit (pentru fișiere executabile: a.out - pentru Linux).
-I cale_către_fișiere_antet	Caută fișiere antet și în calea specificată.
-L cale_către_biblioteci	Caută fișiere bibliotecă și în calea specificată.
-l nume_bibliotecă	Link-editează biblioteca `nume_bibliotecă`. **Atenție!!!** `nume_bibliotecă` nu este întotdeauna același cu numele fișierului antet prin care se include această bibliotecă. Spre exemplu, pentru includerea bibliotecii de funcții matematice, fișierul antet este math.h, iar biblioteca este `m`.
-W tip_w arning	Afișează tipurile de avertismente specificate (Pentru mai multe detalii `man gcc` sau `gcc –help`). Cel mai folosit tip este `all`. Este indicat ca la compilarea cu `-Wall` să nu apară nici un fel de avertismente.
-c	Compilează și asamblează, dar nu link-editează. Generează fișiere obiect, cu extensia `.o`.
-S	Se oprește după faza de compilare, fără să asambleze. Rezultă cod assembler în fișiere cu extensia `.s`.

Despre etapele compilării puteți să citiți mai multe aici [http://elf.cs.pub.ro/so/wiki/laboratoare/laborator-01#fazele-compilarii].

Exemplu

```
gcc -o tema1 tema1.c -lm -Wall
```

Comanda de mai sus are ca efect compilarea și link-editarea fişierului `tema1.c`, cu includerea bibliotecii matematice, afişând toate avertismentele. Fişierul de ieşire se va numi `tema1`.

Atenție!!! Dacă folosiți opțiunea **-o**, nu adăugați imediat după fișierele sursă. Acest lucru ar avea ca efect suprascrierea acestora și pierderea întregului conținut.

Utilitarul Make

Utilitarul `make` determină automat care sunt părțile unui proiect care trebuie recompilate ca urmare a operării unor modificări și declanşează comenzile necesare pentru recompilarea lor. Pentru a putea utiliza `make`, este necesar un fișier de tip `makefile` (numit de obicei `Makefile` sau `makefile`) care descrie relațiile de dependenţă

între diferitele fișiere din care se compune programul şi care specifică regulile de actualizare pentru fiecare fişier în parte.

În mod normal, într-un program, fişierul executabil este actualizat (recompilat) pe baza fișierelor-obiect, care la rândul lor sunt obținute prin compilarea fișierelor sursă. Totuși, acest utilitar poate fi folosit pentru orice proiect care conţine dependenţe şi cu orice compilator/utilitar care poate rula în linia de comandă. Odată creat fișierul `makefile`, de fiecare dată când apare vreo modificare în fișierele sursă, este suficient să rulăm utilitarul `make` pentru ca toate recompilările necesare să fie efectuate. Programul `make` utilizează fișierul Makefile ca bază de date şi pe baza timpilor ultimei modificări a fișierelor din Makefile decide care sunt fișierele care trebuie actualizate. Pentru fiecare din aceste fișiere, sunt executate comenzile precizate in Makefile. În continuare prezentăm un exemplu simplu.

```
# Declaratiile de variabile
CC = gcc
CFLAGS = -Wall -lm
SRC = radical.c
EXE = radical

# Regula de compilare
all:
	$(CC) -o $(EXE) $(SRC) $(CFLAGS)

# Regulile de "curatenie" (se folosesc pentru stergerea fisierelor intermediare si/sau rezultate)
.PHONY : clean
clean :
	rm -f $(EXE) *~
```

Atenție!!! Este obligatorie folosirea tab-urilor (nu spații!). Mai multe informații puteți găsiti pe pagina proiectului [http://www.gnu.org/software/make].

Editoare

Pentru editarea surselor se poate folosi orice editor de text.

Exemple:

- interfața în mod text
  - Vim [http://www.vim.org]
  - Emacs [http://www.gnu.org/software/emacs]
  - nano [http://www.nano-editor.org]
- interfață grafică
  - gedit [http://projects.gnome.org/gedit]
  - Kate [http://kate-editor.org]
- IDE
  - Code::Blocks [http://www.codeblocks.org]

Interacțiunea program-utilizator

Majoritatea algoritmilor presupun introducerea unor date de intrare și calcularea unor rezultate. În cazul programelor de consolă (cele scrise la laborator), datele sunt introduse de la tastatură și afișate pe ecran (alte variante sunt folosirea fișierelor sau preluarea datelor de la un hardware periferic).

Programul dat ca exemplu mai sus folosește funcția de afișare `printf`. Această funcție realizează transferul și conversia de reprezentare a valorii întregi / reale in șir de caractere sub controlul unui format (specificat ca un șir de caractere):

```
printf("format", expr_1, expr_2, ..., expr_n);
```

unde `expr_i` este o expresie care se evaluează la unul din tipurile fundamentale ale limbajului. Este necesar ca pentru fiecare expresie să existe un specificator de format, şi viceversa.

În caz contrar, compilatorul va returna o eroare (în afara cazului în care formatul este obtinut la rulare). Sintaxa unui descriptor de format este:

```
% [ - ] [ Lung ] [ .frac ] [ h|l|L ] descriptor
```

Semnificația câmpurilor din descriptor este descrisă în tabelul următor:

Câmp	Descriere
-	Indică o aliniere la stânga în câmpul de lungime `Lung` (implicit alinierea se face la dreapta).
Lung	Dacă expresia conține mai puțin de `Lung` caractere, ea este precedată de spații sau zerouri, dacă `Lung` începe printr-un zero. Dacă expresia conține mai mult de `Lung` caractere, câmpul de afișare este extins. În absența lui `Lung`, expresia va fi afișată cu atâtea caractere câte conține.
frac	Indică numărul de cifre după virgulă (precizia) cu care se face afișarea.
l	Marchează un `long int`, în timp ce pentru reali `l` determină afișarea unei valori double.
h	Marchează un `short int`.
L	Precede unul din descriptorii `f`, `e`, `E`, `g`, `G` pentru afișarea unei valori de tip `long double`.

Tabelul următor prezintă descriptorii și conversiile care au loc:

Descriptor	Descriere
d	Întreg cu semn în baza 10.
u	Întreg fără semn în baza 10.
o	Întreg fără semn în baza 8.
x sau **X**	Întreg fără semn în baza 16. Se folosesc literele a, b, c, d, e, f mici, respectiv mari.
c	Caracter.
s	Șir de caractere.
f	Real zecimal de forma [-]xxx.yyyyyy (implicit 6 cifre după virgulă)
e sau **E**	Real zecimal în notație exponențială. Se folosește e mic, respectiv E mare.
g	La fel ca și e, E și f dar afișarea se face cu număr minim de cifre zecimale.

Citirea cu format se realizează cu ajutorul funcției `scanf()` astfel:

```
scanf("format", &var_1, &var_2, ..., &var_n);
```

care citește valorile de la intrarea standard în formatul precizat și le depune în variabilele `var_i`, returnând numarul de valori citite.

Atenție!!! Funcția `scanf` primește adresele variabilelor în care are loc citirea. Pentru tipuri fundamentale și/sau structuri, aceasta se obține folosind operatorul de adresă - `&`.

Sintaxa descriptorului de format în acest caz este:

```
% [*] [ Lung ] [ l ] descriptor
```

Semnificația campurilor din descriptor este descrisă în tabelul următor:

Câmp	Descriere
*	Indică faptul că valoarea citită nu se atribuie unei variabile. (valoarea citită poate fi folosită pentru specificarea lungimii câmpului)
	Indică lungimea câmpului din care se face citirea. În cazul în care e nespecificat, citirea are loc până la primul caracter care nu

Lung	face parte din număr, sau până la '\n' (linie nouă/enter).
d	Întreg în baza 10.
o	Întreg în baza 8.
x	Întreg în baza 16.
f	Real.
c	Caracter.
s	Șir de caractere.
L	Indică un întreg long sau un real double.
h	Indică un întreg short.

Pentru scrierea și citirea unui singur caracter, biblioteca `stdio.h` mai definește și funcțiile `getchar()` și `putchar()`:

- `getchar()` are ca efect citirea cu ecou a unui caracter de la terminalul standard. Caracterele introduse de la tastatură sunt puse într-o zonă tampon, până la acționarea tastei `ENTER`, moment în care în zona tampon se introduce caracterul rând nou. Fiecare apel `getchar()` preia următorul caracter din zona tampon.
- `putchar()` afișează caracterul având codul ASCII egal cu valoarea expresiei parametru.

Nota: `getchar()` și `putchar()` nu sunt de fapt funcții, ci niște macroinstrucțiuni definite în `stdio.h`

Pentru citirea și scrierea unei linii biblioteca `stdio.h` definește funcțiile `gets()` și `puts()`:

- `gets(zona)` - introduce de la terminalul standard un șir de caractere terminat prin acționarea tastei `ENTER`. Funcția are ca parametru adresa zonei de memorie în care se introduc caracterele citite. Funcția returnează adresa de început a zonei de memorie; la întalnirea sfarșitului de fișier (CTRL+Z) funcția returnează `NULL`.
- `puts(zona)` - afișează la terminalul standard șirul de caractere din zona dată ca parametru, până la caracterul `null` (`\0`), care va fi înlocuit prin caracterul linie nouă. Funcția returnează codul ultimului caracter din șirul de caractere afișate sau `-1` în caz de eroare.

Exerciții de Laborator

1. Compilați programul din laborator (hello.c [http://ocw.cs.pub.ro/courses/_export/code/programare-ca/laboratoare/lab01?codeblock=0]) utilizând `gcc`.
   - folosiți un fișier de tip `makefile`
   - executabilul se va numi `hello`
2. Într-un director care conține fișierul `hello.c` și **nu** conține niciun fișier de tip `makefile` rulați comanda:

   ```
   make hello
   ```

   Ce observați?

3. Se citește de la tastatură un număr natural în baza 10. Să se afișeze în bazele 8, 10 și 16.
4. Să se calculeze (folosind formule matematice; nu instrucțiuni repetitive) și să se afișeze sumele:

- $S_1 = \sum_{k=0}^{n} k$
- $S_2 = \sum_{k=0}^{n} k^2$

5. Să se determine minimul și maximul a două numere folosind funcția matematică `fabs`.
    - afișați rezultatul cu două zecimale
    - **Atenție!** Trebuie să includeți antetul `math.h` și să compilați cu opțiunea `-lm`