Limbajul C

Initiere in limbajul C

Cuprins

- Caracteristici generale ale limbajului C
- Variabile si expresii aritmetice
- Instructiunea ***for***
- Constante simbolice
- Operatii de intrare-iesire
- Tablouri de date
- Functii C
- Argumentele unei functii – transmiterea argumentelor prin valoare
- Domenii. Variabile externe
- Rezumat

Caracteristici ale limbajului C

- C este un limbaj de programare cu scop general;
- C este un limbaj de nivel mediu; Ce inseamna aceasta?
  - C nu poseda operatii pentru a prelucra direct obiecte compuse (ex. fisiere jpeg);
  - C in sine nu are facilitati de intrare-iesire: nu exista instructiuni READ sau WRITE si nici metode de acces la fisiere, "cablate" in limbaj;
  - C ofera numai structurile fundamentale de control al fluxului: teste, bucle, grupari, si subprograme, insa nu multiprogramare, operatii paralele, sincronizari;
- Avantaje:
  - C poate fi descris intr-un spatiu redus si invatat repede;
  - Este un limbaj independent de arhitectura oricarui calculator si astfel, cu putina grija, este usor a scrie programme portabile (le putem transfera la nivel de cod sursa).

Variabile si expresii aritmetice

- Exemplu: scrieti un program care afiseaza un tabel de temperaturi Fahrenheit si echivalentele lor grade Celsius, folosind formula:

$$C = (5 / 9) * (F - 32)$$

0	-17.8
20	-6.7
40	4.4
60	15.6
...	
260	126.7
280	137.8
300	148.9

Variabile si expresii aritmetice

```
/* Print Fahrenheit-Celsius table
   for f = 0, 20, ..., 300       */
       #include<stdio.h>
       main() {
           int lower, upper, step;
           float fahr, celsius;
           lower = 0; /* lower limit of temperature table */
           upper = 300; /* upper limit */
           step  = 20;  /* step size */
           fahr  = lower;
           while (fahr <= upper) {
              celsius = (5.0 / 9.0) * (fahr - 32.0);
              printf("%4.0f %6.1f\n", fahr, celsius);
              fahr = fahr + step;
           }
        }
```

Variabile si expresii aritmetice

- In limbajul C, toate variabilele trebuie declarate inainte de a fi folosite, inaintea oricarei instructiuni executabile;

```
int lower, upper, step;
float fahr, celsius;
```

- Pe langa tipurile "int" si "float", limbajul C poseda si alte tipuri de date fundamentale :

```
"char"   caracter - un singur octet
"short"  intreg scurt
"long"   intreg lung
"double" numar flotant dubla precizie
```

Variabile si expresii aritmetice

```
lower = 0;
upper = 300;
step  = 20;
fahr  = lower;
```

- Fiecare linie a tabelei este calculata in acelasi fel, vom folosi o bucla care se repeta odata pe linie; acesta este scopul instructiunii "while":

```
while (fahr <= upper) {
     ...
}
```

Variabile si expresii aritmetice

- Temperatura Celsius este calculata si atribuita variabilei "celsius" prin instructiunea:

```
celsius = (5.0 / 9.0) * (fahr - 32.0);
```

- Afisarea rezultatelor se realizeaza astfel:

```
printf("%4.0f %6.1f\n", fahr, celsius);
```

- Fiecare constructie cu % in primul argument al lui "printf" face pereche cu al doilea, al treilea,... argument;
- Aceste perechi trebuie sa corespunda ca numar si tip;

Instructiunea for

- O alta varianta a programului de conversie de temperatura:

```
#include<stdio.h>
main(){ /* Fahrenheit-Celsius table */
      int fahr;
     for (fahr = 0; fahr <= 300; fahr = fahr + 20)
       printf("%4d %6.1f\n", fahr, (5.0 / 9.0) * (fahr - 32));
}
```

- In orice context in care este permisa folosirea valorii unei variabile de un anumit tip, se poate folosi o expresie de acel tip;

Instructiunea **for**

- Instructiunea "***for***" este o bucla, o generalizare a lui "***while***"
- Ea contine trei parti separate prin punct si virgula. Prima parte

```
fahr = 0
```

se executa o data, inainte ca bucla propriu-zisa sa inceapa.

- A doua parte este testul sau conditia care controleaza bucla:

```
fahr <= 300
```

- Se evalueaza aceasta conditie; daca ea este adevarata, este executat corpul buclei (la noi, o singura instructiune "printf").
- Urmeaza apoi pasul de incrementare:

```
fahr = fahr + 20
```

Constante simbolice

- Este o practica proasta aceea de a scrie "numere magice" ca 300 sau 20, intr-un program (aceasta tehnica se numeste hard-coding);
- Cineva care va citi programul mai tarziu va avea dificultati de a intelege logica programului si este greu sa le modificam intr-o maniera sistematica (se pot introduce foarte usor erori);
- Cu ajutorul constructiei "#define", se pot defini la inceputul programului nume sau constante simbolice;
- Compilatorul va inlocui toate aparitiile numelui constantei prin valoarea corespunzatoare;

Constante simbolice

```
#include<stdio.h>
#define LOWER 0     /* lower limit of the table */
#define UPPER 300   /* upper limit */
#define STEP  20    /* step size */
main() {            /* Fahrenheit-Celsius table */
     int fahr;
     for (fahr = LOWER; fahr <= UPPER; fahr = fahr + STEP)
        printf("%4d %6.1f\n", fahr, (5.0 / 9.0) * (fahr - 32));
}
```

Operatii de intrare-iesire

- Vom considera o serie de programe pentru efectuarea de operatii simple asupra datelor alcatuite din caractere;
- Biblioteca standard C poseda functii pentru citirea si scrierea unui caracter;
- Functia *getchar()* citeste urmatorul caracter de intrare de fiecare data cand este apelata si returneaza acel caracter ca valoare. Dupa apelul:

```
c=getchar();
```

variabila "c" contine urmatorul caracter de intrare;

- Functia *putchar(c)* este complementara lui *getchar()*:

```
putchar(c);
```

tipareste continutul variabilei "c" pe un mediu de iesire (monitor);

Operatii de intrare-iesire

- Copiem datele de intrare la iesire:

```
#include<stdio.h>
main() {  /* copy input to output; 1st version */
     int c;
     c = getchar();
     while (c != EOF) {
          putchar(c);
          c = getchar();
     }
}
```

Operatii de intrare-iesire

- Prin conventie, *getchar()* returneaza o valoare care nu este un caracter valid atunci cand intalneste sfarsitul intrarii;
- Am folosit numele simbolic EOF pentru aceasta valoare;

```
#include<stdio.h>
main() { /* copy input to output*/
     int c;
     while ((c = getchar()) != EOF)
          putchar(c);
}
```

Exemple

- Alt exemplu: contorizarea caracterelor;

```
#include<stdio.h>
main() { /* count characters from input */
      long nc;
      nc = 0;
      int c;
      c = getchar();
      while (c!= EOF) {
            ++nc;
            c = getchar();
      }
      printf("%ld\n",nc);
}
```

Exemple

- Contorizarea liniilor

```
#include<stdio.h>
main() { /* contorizarea liniilor in intrare */
      int c, nl;
      nl = 0;
      c = getchar();
      while (c != EOF) {
            if(c == '\n')
                  ++nl;
            c = getchar();
      }
      printf("%d\n", nl);
}
```

Exemple

- Contorizarea de cuvinte

```
#include<stdio.h>
#define YES  1
#define NO   0
main() { /*contorizare linii, cuvinte si caractere la intrare*/
      int c, nl, nw, nc, inword;
      inword = NO;
      nl = nw = nc = 0;
      c = getchar();
      while (c != EOF) {
            ++nc;
            if(c == '\n')
                  ++nl;
            if(c == ' ' || c == '\n' || c == '\t')
                inword = NO;
            else if (inword == NO) {
                  inword = YES;
                  ++nw;
            }
            c = getchar();
      }
      printf("%d %d %d'\n", nl, nw, nc);
}
```

Constante simbolice

```
nl = nw = nc = 0;
nc = (nl = (nw = 0));
```

- Operatorul || inseamna SAU;
- Exemplul foloseste instructiunea "else", care specifica o actiune alternativa ce trebuie executata daca partea de conditie unei instructiuni "if" este falsa.
- Forma generala este:

```
if (expresie)
  instructiune1
else
  instructiune2
```

Tablouri de date

- Vom scrie un program care va contoriza aparitiile fiecarei cifre, a fiecarui caracter de spatiere (blanc, tab, linie noua) si a tuturor celorlalte caractere;
- Exista 12 categorii de intrari;
- Este convenabil sa folosim un tablou pentru a memora numarul de aparitii a fiecarei cifre;

Tablouri de date

```
main(){  /* contorizeaza cifre, spatii albe, alte caractere */
        int c, i, nwhite, nother;
        int ndigit[10];
        nwhite = nother = 0;
        for (i = 0; i < 10; ++i)
                ndigit[i] = 0;
         c = getchar();
        while (c != EOF){
                if (c >= '0' && c <= '9')
                        ++ndigit[c-'0'];
                else if (c == ' ' || c == '\n' || c == '\t')
                        ++nwhite;
                else
                        ++nother;
                c = getchar();
         }
        printf("digits =");
        for (i = 0; i < 10; ++i)
                printf(" %d", ndigit[i]);
        printf("\nwhite space = %d, other = %d\n",
                nwhite, nother);
  }
```

Tablouri de date

- Declaratia:

```
int ndigit[10];
```

Semnifica faptul ca *ndigit* este un tablou de 10 intregi;

- In limbajul C indicii unui tablou intodeauna incep de la zero:

```
ndigit[0],  ndigit[1] ,...,ndigit[9]
```

- Decizia asupra caracterului (daca este o cifra, un caracter de spatiere sau altceva) se ia in secventa:

```
if (c >= '0' && c <= '9')
        ++ndigit[c-'0'];
else if (c == ' ' || c == '\n' || c == '\t')
        ++nwhite;
else
        ++nother;
```

Tablouri de date

- Constructia :

```
if (conditie1)
    instructiune1
else if (conditie2)
    instructiune2
else
    instructiune3
```

apare frecvent in programe ca o modalitate de a exprima deciziile multiple;

Tablouri de date

- Pentru a ilustra folosirea tablourilor de caractere si a functiilor care le manipuleaza, vom scrie un program care citeste un set de linii si o tipareste pe cea mai lunga;
- Pseudocodul:

```
while (mai exista o alta linie)
      if (linia curenta este mai lunga decit linia anterioara)
            salveaza-o pe ea si lungimea ei
tipareste linia cea mai lunga
```

Tablouri de date

- Functia ***getline()*** citeste urmatoarea linie de la intrare;
- Functia ***copy()*** salveaza noua linie intr-un loc sigur;
- Avem nevoie de un program principal care sa controleze functiile ***getline***() si ***copy***();
- Rezultatul este urmatorul:

```
#define MAXLINE 1000  /* lungimea maxima a liniei */
#include<stdio.h>
int getline(char s[], int lim);

main() {                /* gaseste linia cea mai lunga */
        int len;  /* lungimea liniei curente */
        int max;  /* lungimea maxima gasita pina acum */
        char line[MAXLINE]; /* linia curenta introdusa */
        char save[MAXLINE]; /* cea mai lunga linie salvata */
        max = 0;
        while ((len = getline(line, MAXLINE)) > 0)
                if (len > max)  {
                        max = len;
                        copy(line, save);
                }
        if (max > 0)  /* s-a citit cel putin o linie */
                printf("%s", save);
}

int getline (s, lim) { /* citeste linia in s, returneaza lungimea */
        int c, i;
        for(i = 0; i < lim - 1 && (c=getchar())!=EOF && c!='\n';++i)
                s[i] = c;
        if (c == '\n') {
                s[i] = c;
                ++i;
        }
        s[i] = '\0';
        return(i);
}

void copy(char s1[], char[] s2) { /* copiaza pe s1 in s2; s2 suficient de mare */
         int i = 0;
        while ((s2[i] = s1[i]) != '\0')
                ++i;
```

Tablouri de date

- Siruri de caractere in C

```
“hello\n”
-----------------------------
| h | e | l | l | o | \n | \0 |
-----------------------------
```

Functii C

- O functie reprezinta un mod convenabil de a incapsula anumite calcule intr-o “cutie neagra” care poate fi apoi folosita fara sa ne mai intereseze ce se afla inauntrul cutiei;
- Cu functii scrise corect si eficient, este posibil sa ignoram ***"cum"*** este facuta o anumita sarcina;
- Ne este suficient sa stim ***"ce"*** face acea functie;
- Limbajul C este proiectat pentru a face folosirea lor usoara, convenabila si eficienta;
- Am vazut cum se folosesc functii predefinite precum printf() sau getchar();
- Vom scrie acum propriile noastre functii!

Functii C

- Vom scrie o functie ***"power(m,n)"*** care va ridica un intreg ***m*** la o putere intreaga pozitiva ***n***;

```
#include <stdio.h>

int power(int x, int n);

main() { /* testeaza functia power */
	int i;
	for (i = 0; i < 10; ++i)
		printf ('%d %d %d\n', i, power(2,i), power(-3,i));
}

int power(int x,int n) { /* ridica pe x la puterea a n-a ; n > 0 */
	int i, p;
	p = 1;
	for (i = 1; i <= n; ++i)
		p = p * x;
	return(p);
}
```

Functii C

- Functia *power* este apelata de doua ori in linia:

```
printf("%d %d %d\n", i, power(2,i), power(-3,i));
```

- Fiecare apel trimite doua argumente lui *power*, care returneaza un numar intreg care trebuie formatat si afisat;
- ***Declaratia*** functiei *power()* este inaintea functiei *main()* si contine:
  - Argumentele functiei cu tipul lor;
  - Tipul valorii calculate si returnat de functie (in cazul nostru int);
- ***Definitia*** functiei este imediat dupa functia *main()*;

Argumentele unei functii – transmiterea argumentelor prin valoare

- In C, toate argumentele functiei sunt transmise “prin valoare” ;
- Aceasta inseamna ca functiei apelate i se transmit valorile argumentelor in variabile temporare;
- Functia apelata nu poate altera o variabila in functia apelanta!

```
int power(int x,int n) { /*ridica pe x la puterea n; n > 0*/
      int p;
      for (p = 1; n > 0; --n)
            p = p * x;
      return(p);
}
```

- Argumentul ***n*** este folosit ca o variabila temporara si este decrementat pana cand devine zero, nu mai este nevoie de variabila ***i***.
- Ceea ce se face cu ***n*** in interiorul lui ***power*** nu are nici un efect asupra argumentului cu care a fost apelata ***power*** initial;

Argumentele unei functii – transmiterea argumentelor prin valoare

- Cand este necesar, este posibil sa facem ca o functie sa modifice o variabila in rutina apelanta;
- Apelantul trebuie sa dea ***adresa*** variabilei de setat (in mod tehnic, sa creeze un pointer la variabila),iar functia apelata trebuie sa declare argumentul ca fiind un pointer si sa refere variabila reala in mod indirect prin el;

Domeniu. Variabile externe

- Variabilele din functia *main* (line, save, etc) sunt private sau locale lui main;
- Deoarece ele sunt declarate in *main*, nici o alta functie nu poate avea acces direct la ele;
- Fiecare variabila locala dintr-o rutina se ***“creaza”*** numai atunci cand functia este apelata si ***"dispare"*** cand functia isi termina activitatea;
- Acestea sunt variabile ***“automate”***;
- Ca o alternativa la variabilele automate, este posibil sa definim variabile care sa fie "externe" tuturor functiilor, adica, variabilele globale care sa fie accesate prin nume de orice functie care doreste sa o faca;

Domeniu. Variabile externe

- O variabila externa trebuie sa fie definita in afara oricarei functii; acest lucru face sa se aloce memorie reala pentru ea;
- Variabila trebuie apoi declarata in fiecare functie care vrea sa o foloseasca;
- Aceasta se poate face fie printr-o declaratie explicita “***extern***” ;
- Vom rescrie programul precedent, in care ***line***, ***save***, ***max*** vor fi declarate variabile externe;

```
#define MAXLINE 1000 /* marimea maxima a liniei de intrare */
#include<stdio.h>
char line[MAXLINE];  /* linia de intrare */
char save[MAXLINE];  /* cea mai lunga linie este salvata aici*/
int max; /* lungimea liniei celei mai mari */
void main()   /* gaseste linia cea mai lunga ;versiune specializata*/
{
        int len;
        extern int max;
        extern char save[];
        max = 0;
        while ((len = getline() > 0)
                if (len > max)  {
                        max = len;
                        copy();
                }
        if (max > 0)     /* a fost cel putin o linie */
        printf("%s", save);
}

void getline()              /* versiune specializata */
{
        int c, i;
        extern char line[];
        for (i = 0; i < MAXLINE-1 && (c = getchar()) != EOF && c != '\n'; ++i)
                line[i] = c;
        if (c == '\n') {
                line[i] = c;
                ++i;
        }
        line[i] = '\0';
        return(i);
}

void copy()                         /* versiune specializata */
{
        int i;
        extern char line[], save[];
        i = 0;
        while ((save[i] = line[i]) != '\0')
                ++i;
}
```

- Atentie la cuvintele ***"declaratie"*** si ***"definitie"*** cand ne-am referit la variabile externe in aceasta sectiune;
- ***"Definitiile"*** se refera la locul in care variabila este efectiv creata si i se asigneaza memorie;
- ***"Declaratie"*** se refera la locul unde natura variabilei este declarata dar nu i se aloca memorie;

Rezumat

- Am acoperit ceea ce, conventional, poate fi numit esenta lui C;
- Cu aceste cateva notiuni fundamentale, puteti sa scrieti programe utile de marime considerabila!

Video tutorials

- https://www.youtube.com/watch?v=ot9LoXNSqLU
- https://www.youtube.com/watch?v=-CpG3oATGIs
- https://www.youtube.com/watch?v=YOLN-t09-tM

Medii integrate de dezvoltare C

- Microsoft Visual Studio for Community 2017
- Code::Blocks
- Eclipse