Laboratorul 1. NOȚIUNI INTRODUCTIVE PENTRU LIMBAJELE DE PROGRAMARE C/C++

1.1. Etapele parcurse pentru obținerea unui program executabil

Calculatorul prelucrează datele de intrare printr-o succesiune de transformări pentru a obține datele de ieşire. Succesiunea de prelucrări este descrisă prin program. Fiecare prelucrare este rezultatul uneia sau mai multor instrucțiuni. La scriere unui program se are în vedere ordinea de efectuare a prelucrărilor, ea respectând un anumit algoritm.

Principalele etape care se parcurg pentru obținerea unui program executabil sunt (Fig.1.1):

- **etapa de scriere a programului sursă** (fişier cu extensia **.cpp**): este etapa în care, prin intermediul unui editor de text sau IDE (Integrated Development Environment – mediu de dezvoltare a programelor), se scrie programul, respectându-se sintaxa limbajului şi logica aplicației;
- **etapa de compilare**: mai întâi se execută directivele de preprocesare prin expandarea în codul sursă şi apoi se transpune programul sursă în program obiect (fişier cu extensia .obj);
- **etapa de link-editare**: crearea legăturilor între modulele obiect obținute la pasul anterior şi cu bibliotecile de sistem şi transformarea intr-un program executabil (fişier cu extensia .exe);
- **etapa de lansare în execuție** a fişierului executabil.

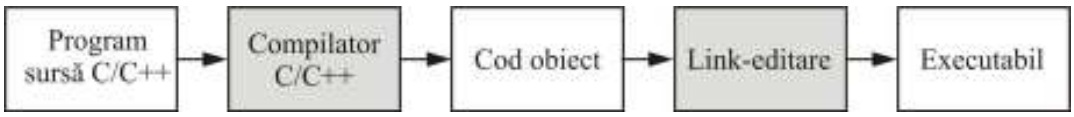


Fig.1.1. Principalele etape parcurse pentru obținerea unui program executabil

1.2. Elemente de bază ale limbajelor C/C++

La scrierea unui program în C/C++ se folosesc unități sintactice care se încadrează în diferite categorii, cum ar fi:

- **identificatori** – nume care desemnează tipuri de date, constante, variabile, funcții, etc.
- **semne de punctuație şi caractere speciale** – simboluri care separă diverse entități ale programului, unii pot avea dublă semnificație;
- **spații albe** – sunt caractere cu rol de delimitare sau care cresc lizibilitatea programului sursă;
- **operatori** – simboluri utilizate pentru precizarea operațiilor ce se efectuează;
- **constante** – valori fixe reprezentând valori numerice, întregi sau reale, caractere sau şiruri de caractere.

1.2.1. Identificatori

Identificatorii limbajului C++, ca şi în limbajul C standard, sunt formați cu ajutorul caracterelor alfanumerice şi liniuța de subliniere (**underscore**), "_". Aceştia denumesc tipuri de date, variabile, funcții, operatori, instrucțiuni etc.

Primul caracter al unui identificator nu poate fi o cifră.

Exemple

```
Raza        // valid
raza        // valid
mesaj       // valid
_maxx       // valid
a5          // valid
5a          // invalid, primul caracter este cifră
A_B_C       // valid
A%B         // invalid, caracterul % nu poate fi folosit într-un identificator
```

Limbajul C/C++ este **case sensitive**, adică face diferență între majuscule şi minuscule, astfel, în exemplele date anterior, numele *Raza* este diferit de *raza*.

Din mulțimea identificatorilor posibili, se remarcă **cuvintele cheie** (**reserved keywords**) ale limbajului, identificatori a căror semnificație este predefinită şi nu poate fi modificată de programator. Cuvintele cheie ale limbajului C++ sunt prezentate în Tabelul nr. 1.1.

Tabelul nr.1.1. Cuvinte cheie ale limbajului C++

Cuvinte cheie ale limbajului C++					
_asm	asm	auto	break	case	_cdecl
cdecl	char	class	const	continue	_cs
default	delete	do	double	_ds	else
enum	_es	_export	extern	_far	far
_fastcall	float	for	friend	goto	_huge
huge	if	inline	Int	_interrupt	interrupt
_loadds	long	_near	Near	new	operator
_pascal	pascal	private	protected	public	register
return	_saveregs	_seg	short	signed	sizeof
_ss	static	struct	switch	template	this
typedef	union	unsigned	virtual	void	volatile
while					

1.2.2. Semne de punctuație şi caractere speciale

Semne de punctuație şi caractere speciale constituie un set de caractere cu diferite utilizări, care în funcție de poziția ocupată în textul programului sursă pot determina acțiunile care vor fi executate.

Aceste caractere sunt:

[ ] ( ) { } * , : = ; ... #

Unele caractere pot desemna operatori (ca de exemplu [ ] * , =) altele nu desemnează o

operație, ci delimitează zone în alcătuirea codului sursă. Spre exemplu, caracterul ; (punct şi virgulă) marchează sfârşitul unei instrucțiuni, iar { şi } (paranteze acolade) marchează începutul şi respectiv sfârşitul unui bloc de instrucțiuni.

Caracterul # însoțeşte numai directivele de preprocesare.

Semnificația şi modul de utilizare al acestor caractere se va preciza în capitolele următoare.

1.2.3. Spații albe

Spațiile albe sunt caractere speciale care, deşi nu sunt afişabile, produc efecte în afişarea textului.

Aceste caractere sunt: space (are valoarea 32 în codul ASCII), tab (are valoarea 9 în codul ASCII), newline (are valoarea 10 codul ASCII), vertical-tab (are valoarea 11 în codul ASCII), formfeed (are valoarea 12 în codul ASCII), carriage-return (are valoarea 13 în codul ASCII).

Aceste caractere sunt interpretate ca separatori când sunt plasate între entități diferite ale programului sau ca şi caractere dacă intră în alcătuirea şirurilor de caractere.

Sunt situații în care aceste caractere se folosesc pentru a creşte lizibilitatea programului sursă, caz în care ele vor fi ignorate de compilator.

1.3. Structura programului C/C++

Acțiunile efectuate de un program C/C++ se pot grupa respectând algoritmul transpus, programul fiind structurat în module, fiecare având un rol bine definit.

Fiecare acțiune este determinată de o instrucțiune.

O succesiune de instrucțiuni poate fi grupată într-un bloc numit funcție.

Structura generală a unui program este alcătuită, în general, din 3 zone:

I < directive de preprocesare >

II < declarații globale >

III definiții de funcții

Zonele I şi II sunt opționale, dar cea de a III-a este obligatoriu să existe.

Preprocesorul efectuează operații înainte de compilarea codului, indicate prin aşa numite directive de preprocesare, marcate totdeauna de caracterul #. Ele produc modificări în codul care va fi compilat, cum ar fi inserarea unor fişiere fie din biblioteca standard, fie create de utilizator, declararea de macrodefiniții, etc.

Declarațiile globale pot conține definiții de tipuri de date, prototipuri de funcții, declarații de variabile globale.

Programul trebuie să conțină cel puțin o funcție, denumită **main()**, execuția programului începând cu această funcție. **Funcția main() este unică, ea nu poate fi supradefinită.**

Restul funcțiilor pot efectua diferite acțiuni, ele putând fi definite de utilizator, sau să fie preluate din bibliotecile de funcții C/C++.

Funcțiile în C/C++, spre deosebire de alte limbaje, cum ar fi Pascal, nu pot include definirea altor funcții. Deci funcțiile sunt definite în afara oricărei funcții.

Definiția unei funcții are următoarea structură:

```
<tip_r> nume_functie (< lista_parametri >)
{
        <declaratii locale >
        secventa de instructiuni
}
```

tip_r – precizează tipul rezultatului funcției. Dacă nu este specificat, în mod implicit rezultatul întors de funcție este o valoare numerică întreagă, tipul de dată fiind int. Dacă funcția nu returnează nici o valoare, acest lucru este precizat prin tipul de dată **void**.

Dacă funcția prelucrează date existente în momentul apelului funcției (date de intrare), aceste date pot fi preluate prin lista de parametri (lista_parametri). Aceasta are următoarea sintaxă:

tip_p nume_parametru< , tip_p nume_parametru ,…, tip_p nume_parametru >

unde ***tip_p*** reprezintă tipul parametrului, iar ***nume_parametru*** un identificator diferit de cuvintele cheie sau cei care au deja o semnificație în domeniul funcției.

Lista parametrilor este încadrată între paranteze, ( ).

În cazul în care nu sunt necesari parametri, acest fapt este precizat prin cuvântul cheie **void** care desemnează lipsa parametrilor. În lipsa oricărei specificări pentru lista de parametri, în mod implicit compilatorul interpretează că lista este void.

Apelul funcției (lansarea în execuție a funcției) se face prin precizarea numelui funcției, însoțit de paranteze rotunde în interiorul cărora se specifică valorile atribuite parametrilor, numite valori efective, dacă aceştia există:

nume_functie(<valoare_parametru1, valoare_parametru2,…, valoare_parametrun>);

Corpul funcției este format din declarații locale şi dintr-o succesiune de instrucțiuni cuprinse între acolade, { }.

Declarațiile locale, făcute în interiorul unei funcții, pot fi folosite numai în interiorul acelei funcții, ele încetând să mai existe odată cu terminarea execuției funcției.

Câteva exemple de definiții de funcții:

1.	void functia_1(void)
	{
	printf("Se executa functia 1");
	}
		- funcția nu preia parametri şi nu întoarce nici un rezultat.

2.	void functia_2(int parametru)
	{
	printf("Se executa functia 2");
	}
		- funcția preia un parametru şi nu întoarce nici un rezultat.

3.	int functia_3(int parametru_1, int parametru_2)
	{
	printf("Se executa functia 3");
	return (parametru_1+parametru_2);
	}
		- funcția preia doi parametri şi întoarce un rezultat.

Apelul (lansarea în execuție) acestor funcții se face astfel:

1.	functia_1();	- apelul funcției fără parametri.
2.	functia_2(100);	- apelul funcției cu un parametru; valoarea preluată de parametru este 100.
3.	functia_3(100, 500);	- apelul funcției cu doi parametri; valorile preluate de parametri sunt 100 şi respectiv 500; funcția întoarce valoare, 600 (100+500), valoare ce poate fi preluată de o variabilă sau folosită într-o expresie.

1.4. Directive de preprocesare

Preprocesorul efectuează operații prealabile compilării, indicate de directivele de preprocesare. Acestea sunt precedate de caracterul #.

În mod frecvent sunt utilizate directivele **#define** şi **#include**.

Directiva # define

Directiva **define** este utilizată pentru a asocia un identificator cu un şir de caractere. În etapa de precompilare fiecare apariție a identificatorului va fi înlocuită, caracter cu caracter, cu şirul corespunzător. Forma generală de utilizare a directivei #define este:

#define nume sir de caractere

Exemple

```
#define TRUE   1
#define FALSE  0
#define NULL   0
#define MIN    10
#define MAX  MIN+100
#define f(a, b) a+b
```

Directiva #define :

- permite folosirea constantelor simbolice în programe
- uzual, dar nu obligatoriu, simbolurile sunt scrise cu litere mari
- nu se termină cu ';'
- apar în general la începutul unui fişier sursă
- fiecare apariție a simbolului este înlocuită, caracter cu caracter, cu valoarea sa

Directiva #include

Directiva de preprocesare **#include** determină includerea unui fişier sursă în alt fişier sursă. Sintaxa de utilizare are una din formele:

#include < nume_fisier >

#include " nume_fisier "

Pentru prima variantă, fişierul va fi căutat într-un director predefinit care conține fişiere header puse la dispoziție de mediul de programare, cea de a doua va căuta fişierul pe calea specificată în numele fişierului. Dacă nu e precizată calea fişierului, va fi căutat în directorul curent.

Exemple de folosire a directivei #include

```
#include <stdio.h>                  // fişierul e căutat în directorul predefinit
#include “c:\dir1\dir2\fis_meu1.h”  // fişierul este căutat pe calea  c:\dir1\dir2\
#include “fis_meu2.h”               // fişierul este căutat în directorul curent
```

1.5. Primul program C/C++

În continuare se prezintă un exemplul de program, exemplu consacrat ca fiind primul program în mare parte a manualelor de specialitate, fiind urmat de comentarii referitoare la noțiunile prezentate până la acest moment.

Exemplu

```
1   #include <stdio.h>
2
3   int main(int argc, char* argv[])
4   {
5       printf("Hello World!\n");
6       return 0;
7   }
```

La execuția programului se va afişa: Hello World!

1.6. Utilizarea comentariilor

La scrierea programelor sursă este util în numeroase situații să se introducă anumite comentarii care să explice eventuale secvențe de instrucțiuni. Comentariile sunt ignorate de compilator putând conține orice text, ele fiind utile doar programatorului.

Comentariile se pot scrie astfel:

- comentarii pe un rând – sunt precedate de caracterele //
- comentarii pe mai multe rânduri – sunt marcate între caracterele /* şi */

Exemplu

```
1   //exemplificarea utilizării comentariilor
2   /*
3       Acesta este un comentariu scris pe mai multe linii.
4       Comentariile sunt ignorate de compilator.
5   */
6   #include <stdio.h>
7
8   int main()
9   {
10      // comentariu pe o linie
11      printf("Hello World !\n");
12      printf("Hello World !\n");
```

```
13          return 0;              // instrucțiunea return încheie execuția unei funcții
14          printf("Hello World !\n");      /* aceasta linie nu se va executa
15                                             deoarece se afla după instrucțiunea
16                                              return */
17      }
```

1.7. Tipuri de date fundamentale

Tipul unei date determină dimensiunea zonei de memorie ocupată şi valorile pe care le poate lua. Tipurile datelor se pot grupa în:

- **tipuri fundamentale;**
- **tipuri derivate.**

Tipurile de date fundamentale cuprind **tipurile aritmetice de bază** şi **tipul void**.

Există patru tipuri aritmetice de bază, specificate prin cuvintele cheie: **char**, **int**, **float** şi **double**. Gama de valori poate fi extinsă prin modificatori de tip desemnați prin cuvintele cheie: **signed**, **unsigned**, **short**, **long**. Tipurile întregi ce se obțin prin combinarea tipurilor de bază cu modificatorii de tip sunt prezentate în Tabelul nr. 1.2, iar cele reprezentate în virgulă mobilă în Tabelul nr. 3.3.

Tipul fundamental void indică absența oricărei valori şi se utilizează în următoarele situații: declarația unei funcții fără parametri sau care nu returnează un rezultat, tipul pointer generic şi conversii de tip cu operatorul cast pentru pointeri.

Pentru determinarea numărului de octeți de memorie folosiți pentru stocarea datelor se poate folosi operatorul **sizeof (**va fi prezentat în capitolul destinat operatorilor).

Iată câteva exemple de expresii care returnează ca valoare spațiul de memorie ocupat de date de diferite tipuri:

```
sizeof (long int);              // expresia returnează valoarea 4
sizeof (unsigned char)          // expresia returnează valoarea 1
sizeof (long double)            // expresia returnează valoarea 8 sau 10
```

Observații

- Modul de memorare al acestor tipuri de date depinde de varianta de implementare a limbajului C/C++. De exemplu, tipul int este reprezentat pe 2 byte în unele versiuni şi pe 4 byte în altele. În plus, modul de implementare poate fi modificat prin utilizarea modificatorilor :

  signed - cu semn (pentru *char, int, long int*)
  unsigned - fără semn (pentru *char, int, long int*)
  short - scurt (pentru *int*)
  long - lung (pentru *int* şi *double*)

Tabelul nr. 1.2. Tipuri de date întregi

Tip	Spațiu de memorie ocupat	Domeniu de valori
char = signed char	8 biți	-128…127
unsigned char	8 biți	0…255

short int = signed short int	16 biți	-32768…32767
int = signed int	16/32 biți	-32768…32767 / -2147483648…2147483647
unsigned int = unsigned short int	16/32 biți	0…65535 / 0…4294967295
long int = signed long int	32 biți	-2147483648…2147483647
unsigned long int	32 biți	0…4294967295

Tabelul nr.1.3. Tipuri de date în virgulă mobilă

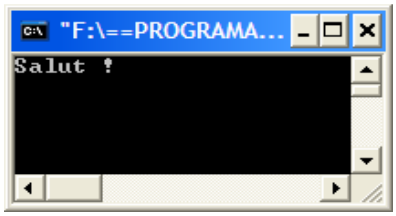
Tip	Spațiu de memorie ocupat	Domeniu de valori
float	32 biți	+/-(3.4E-38…3.4E+38) precizie 7 cifre
double	64 biți	+/-(1.7E-308…1.7E+308) precizie 15 cifre
long double	80 biți	+/-(3.4E-4932…1.1E4932) precizie 19 cifre

Exemple

```
/*************************************************************************
Exemplul 1.1. - Afişarea unor mesaje
*************************************************************************/
#include <stdio.h>              //directiva de preprocesare  - in fişierul stdio.h este definita
                                // functia printf()- functie de afisare

void main(void)                 //orice program C/C++ trebuie să conţină o funcţie main() - cu
                                // ea  începe execuția programului; funcția main() este unică
{
        printf("Salut !");      //functia printf() - afiseaza pe ecran sirul de caractere "Salut !"
                                //şirurile de caractere se încadrează între ghilimele
}
/*************************************************************************/
```

La execuția programului se afişează:

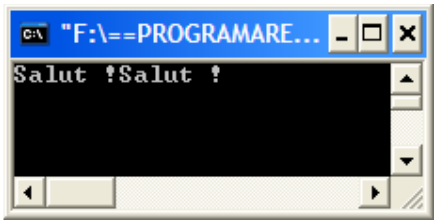



```
/*************************************************************************
Exemplu1 1.2. - Afişarea unor mesaje
*************************************************************************/
#include <stdio.h>

void main(void)
{
        printf("Salut !");      //functia printf() - afiseaza pe ecran sirul de caractere "Salut !"
        printf("Salut !");      //functia printf() - afiseaza pe ecran sirul de caractere "Salut !"
```

```
}
/*************************************************************************/
```

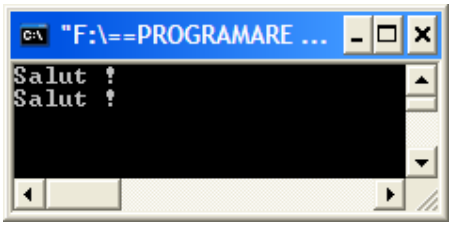
La execuția programului se afișează:




```
/*************************************************************************
Exemplu1 1.3. - Afişarea unor mesaje
*************************************************************************/
#include <stdio.h>

void main(void)
{       printf("Salut !");
        printf("\nSalut !");          // secvența \n inclusă în şirul de caracter are ca efect
                                      // mutarea cursorului la începutul rândului următor
}
/*************************************************************************/
```

La execuția programului se afișează:

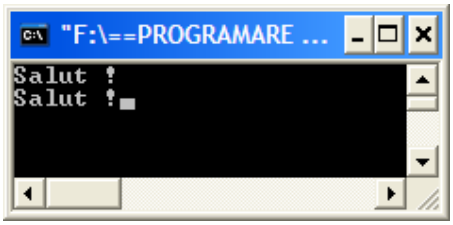



```
*************************************************************************
Exemplu1 1.4. - Afişarea unor mesaje
*************************************************************************/
#include <stdio.h>
#include <conio.h>                    // in fişierul conio.h este definita functia getch() - functie
                                      // care citeşte de la tastatură un caracter
void main(void)
{       printf("Salut !");
        printf("\nSalut !");
        getch();                      //functia getch() citeşte de la tastatură un caracter

        /* caracterul citit este ignorat, funcția fiind apelată pentru a întrerupe execuția programului
        până la apăsarea unei taste; în absența ei, programul se execută, dar utilizatorul nu poate
        vizualiza mesajele afişate datorită faptului că, odată cu terminarea programului se închide şi
        fereastra în care au fost afişate mesajele. */
}
/*************************************************************************/
```

La execuția programului se afișează:

(fereastra se închide după apăsarea unei taste)

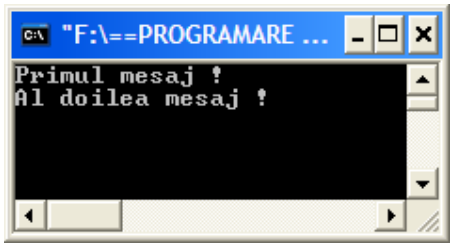



```
/*************************************************************************
Exemplu1 1.5. – Utilizarea comentariilor
*************************************************************************/
/*      Acesta este un comentariu scris pe mai multe linii.
        Comentariile sunt ignorate de compilator
*/
#include <stdio.h>
#include <conio.h>

void main()
{       // comentariu pe o linie
        printf("Primul mesaj !\n");
        printf("Al doilea mesaj !\n");
        getch();
        return;                         // instrucțiunea return încheie execuția functiei main()

        printf("Al treilea mesaj !\n");         /* aceasta linie nu se va executa deoarece
                                                   se afla după instrucțiunea return*/
}
/*************************************************************************/
```

La execuția programului se afișează:



- comentariile sunt ignorate de compilator, deci ele nu produc nici un efect la execuția programului;
- instrucțiunile aflate după instrucțiunea return (instrucțiunea return încheie execuția funcției main, deci și a programului) nu sunt executate, astfel că șirul "Al treilea mesaj !\n" nu este afișat pe ecran.

Întrebări. Exerciții. Probleme.

1. Descrieți componentele și rolul sistemului de operare.

2. Scrieți un program care să afișeze următorul text:

   Program EXEMPLU.

   Limbajul de programare folosit: C++

   Autor: studentul ....

3. Scrieți un program care să afișeze un caracter, un șir de caractere, o valoare numerică întreagă și o valoare numerică reală.