Evoluția limbajelor de programare.

I Cod mașină
- programul în format binar, executat direct de procesor

II Limbaj de asamblare
- instrucțiunile în format binar sunt înlocuite cu mnemonice, etichete simbolice pentru acces la memorie

III Procedural
- descompune programul în proceduri / funcții

IV Modular
- descompune programul în module

V Orientat Obiect
- descompune programul într-o mulțime de obiecte care interacționează

Programarea orientată-obiect oferă o abstractizare puternică, unde programatorul poate exprima soluția mai natural. Programul este descompus într-un set de obiecte, iar obiectele interacționează

pentru a rezolva problema. Tipuri noi de date modelează elemente din spațiul problemei.

Un obiect este o entitate care:

- are o stare
- poate executa anumite operații
- este o combinație de date + metode

C/C++ Limbaj compilat

Programul C/C++ trebuie compilat pentru a putea fi executat. Programul scris în fișiere text trebuie transformat în cod binar ce poate fi executat de procesor.

Fișiere sursă - fișiru text ce conțin programul scris într-un limbaj de programare

compilatorul - analizează fișierele și creează fișiere obiect

Fișiere obiect - fișiru intermediar, conține bucăți incomplete din programul final

Linker - combină fișierele obj și creează programul care poate fi executat

↓

Executabil

↓ Sistemul de operare încarcă fișierul în memorie și execută programul

↓

Program în memorie

Elemente de bază

Identificator:

- nume (secvență de litere / cifre) pentru elemente din program

Cuvinte rezervate (keywords):

- identificatori cu semantică specială pentru compilator
- int, if, for, ...

Literals:

- constante specificate direct în codul sursă
- "Hello", 72, 4.6, 'c'

Operatori:

- aritmetici, pe biți, relaționali

- +, -, <<, ...

Separatori:

- semne de punctuație folosite pentru a defini structura
- {}, ()

Whitespace:

- caractere ignorate de compilator
- space, tab, linie nouă

Comentarii:

- // this is a single line comment
- /* this is a
  * multiline
  */ comment
- sunt ignorate de compilator

Tipuri de date

Name	Description	Size	Range
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	Wide character.	2 *or* 4 bytes	1 wide character

Vectori

Dacă T este un tip de dată:

- T[n] este un vector cu n elemente de tip T
- indicii sunt de la 0 la n-1
- operatorul de indexare: []
- vector multidimensional: t[n][m]

```c
#include <stdio.h>

int main() {
      int a[5]; // create an array with 5 elements
      a[0] = 1; //index start from 0
      a[1] = 2;
      printf("a[0]=%d \n", a[0]);
      printf("a[1]=%d \n", a[1]);
        //!!! a[2] uninitialised
      printf("a[2]=%d \n", a[2]);

      int b[] = { 1, 2, 3, 5, 7 };
      printf("b[0]=%d \n", b[0]);
      b[0] = 10;
      printf("b[0]=%d \n", b[0]);
      return 0;
}
```

C Stringuri

- Sunt reprezentate ca un vector char, ultimul caracter este '\0'
- C are un modul <string.h> pentru a manipula stringuri

strlen - returnează numărul de caractere

strcpy - Copiază caractere de la sursă în destinație

strcmp - compară stringurile

strcat - concatenează sursa cu destinația

Obs: nici o metodă nu verifică dacă ∃ suficient loc pt. operație.

```c
#include <stdio.h>
#include <string.h>

int main() {
	char name[100];

	strcpy(name, "Popescu");

	printf("name:%s l=%d", name, strlen(name));

	char name2[100];

	strcpy(name2, name);
	printf("name:%s l=%d", name2, strlen(name2));

	return 0;
}
```

Structuri

- este o colecție de elemente de tipuri diferite
- permite gruparea diferitelor tipuri de date simple într-o structură

```c
struct name{
  type1 field1;
  type2 field2
}
```

```c
struct car{
  int year;
  int nrKm;
}
```

```c
car c;
c.year = 2010
c.nrKm = 30000;
```

```c
#include <stdio.h>
//introduce a new struct called Car
typedef struct {
      int year;
      int km;
} Car;

int main() {
      Car car, car2;

      //initialise fields
      car.year = 2001;
      car.km = 20000;

      printf("Car 1 fabricated:%d Km:%d \n", car.year, car.km);

      //!!! car2 fields are uninitialised
      printf("Car 1 fabricated:%d Km:%d \n", car2.year, car2.km);
      return 0;
}
```

Pointeri

- Pointer este un tip de date special, folosit pentru a lucra cu adrese de memorie.
- Poate stoca adresa unei variabile.

Declarare:

- Se pune * înainte de numele variabilei, de exemplu: int *a, long *a, char *a

Operatori:

& - returnează adresa unde este stocată valoarea unei variabile

* - returnează valoarea stocată în locația de memorie specificată

```c
#include <stdio.h>

int main() {
	int a = 7;
	int *pa;

	printf("Value of a:%d address of a:%p \n", a, &a);
	//assign the address of a to pa
	pa = &a;
	printf("Value of pa:%d address of pa:%p \n", *pa, pa);

	//a and pa refers to the same memory location
	a = 10;
	printf("Value of pa:%d address of pa:%p \n", *pa, pa);
	return 0;
}
```

Instrucțiuni

Toate se termină cu ;

if, if-else, else if

```
if (condition){
   //statements executed only if the condition is true
}
```

```
if (condition){
   //statements executed if the condition is true
} else {
  //statements executed only if the condition is not true
}
```

```
if (condition1){
   //statements executed if the condition1 is true
} else if (condition2){
  //statements executed only if condition1 is not true and the condition2 is true
}
```

switch-case

```
switch(expression)
{
  case constant1:
       statementA1
       statementA2
       ...
   break;
   case constant2:
      statementB1
      statementB2
       ...
      break;
    ...
  default:
     statementZ1
     statementZ2
     ...
}
```

while , do-while

```
while(condition)
{
  statement1
  statement2
  ...
}
```

```
do
{
  statement1
  statement2
  ...
}
while(condition);
```

for

```
for(initialization; condition; incrementation)
{
  //body
}
```

Citire / Scriere

printf() - tipărește în consolă

```
#include <stdio.h>

int main() {
	int nr = 5;
	float nrf = 3.14;
	char c = 's';
	char str[] = "abc";

	printf("%d  %f  %c %s", nr, nrf , c, str);
	return 0;
}
```

scanf() - citește de la tastatură; returnează nr. negativ în cazul de eroare

```
int main() {
	int nr;
	float f;
	printf("Enter a decimal number:");
	//read from the command line and store the value in nr
	scanf("%d", &nr);
	printf("The number is:%d \n", nr);

	printf("Enter a float:");
	if (scanf("%f", &f) == 0) {
		printf("Error: Not a float:");
	} else {
		printf("The number is:%f", f);
	}
	//wait until user enters 'e'
	while(getchar()!='e');
	return 0;
}
```

Format specifiers

- %d - număr întreg
- %c - caracter
- %f - float
- %s - cstring

%p - pointer

Dacă se citește un string se citește doar până la primul spațiu. Se poate folosi gets pentru a citi o linie.

Specificații

- nume sugestiv
- scurtă descriere a funcției (ce face)
- semnificația parametrilor
- condiții asupra parametrilor (precondiții)
- ce se returnează
- condiții care sunt satisfăcute după execuția funcției (postcondiții)

```
/*
* Verify if a number is prime
* nr - a number, nr>0
* return !=0 if the number is prime (1 and nr are the only dividers)
*/
int isPrime(int nr);
```