Tablouri si Pointeri

*Acest **curs** prezinta **Tablouri si Pointeri**.*
In acest PDF poti vizualiza cuprinsul si bibliografia (daca sunt disponibile) si aproximativ doua pagini din documentul original.
*Arhiva completa de pe site contine un fisier, intr-un numar total de **11 pagini**.*
Fisierele documentului original au urmatoarele extensii: doc.

Extras

2. Pointeri.

Un pointer este o variabila care are ca valori adrese ale altor variabile, sau mai general adrese de memorie.

Un pointer este asociat unui tip de variabile, deci avem pointeri catre int, char, float, etc.

În general o variabila pointer p catre tipul T se declara:

T *p;

Un tip pointer la tipul T are tipul T*.

Exemple:

int j,*pj; /*pj este o variabila de tip pointer la întregi*/

char c, *pc;

Se introduc doi noi operatori:

• operatorul de adresare & - aplicat unei variabile furnizeaza adresa acelei variabile

pj=&j; /* initializare pointer */

pc=&c;

Aceste initializari pot fi facute la definirea variabilelor pointeri:

int j, *pj=&j;

char c, *pc=&c;

O greseala frevent comisa o reprezinta utilizarea unor pointeri neinitializati.

int *px;

px=5; / gresit, pointerul px nu este initializat (legat

la o adresa de variabila întreaga */

Pentru a evita aceasta eroare vom initializa în mod explicit un pointer la NULL, atunci când nu este folosit.

• operatorul de indirectare (dereferentiere) * – permite accesul la o variabila prin intermediul unui pointer. Daca p este un pointer de tip T*, atunci *p este obiectul de tip T aflat la adresa p.

În mod evident avem:

*(&x) = x;

&(*p) = p;

Exemplu;

int *px, x;

x=100;

px=&x; // px contine adresa lui x

printf(“%dn”, px); // se afiseaza 100

Dereferentierea unui pointer neinitializat sau având valoarea NULL conduce la o eroare la executie.

3. Pointeri void.

Pentru a utiliza un pointer cu mai multe tipuri de date, la declararea lui nu îl legam de un anumit tip.

void *px; // pointerul px nu este legat de nici un tip

Un pointer nelegat de un tip nu poate fi dereferentiat.

Utilizarea acestui tip presupune conversii explicite de tip (cast). Exemplu:

int i;

void *p;

. . .

p=&i;

*(int)p=5; // ar fi fost gresit *p=5

Exemplul 17: Definiti o functie care afiseaza o valoare ce poate apartine unuia din tipurile: char, int, double.

#include <stdio.h>

enum tip {caracter, intreg, real};

void afisare(void *px, tip t) {

switch(t) {

case caracter:

printf("%cn", *(char*)px); break;

case intreg:

printf("%dn", *(int*)px); break;

```
case real:

printf("%lfn",*(double*)px); break;

}

}


void main(void) {

char c='X';

int i=10;

double d=2.5;

afisare(&c, caracter);

afisare(&i, intreg);

afisare(&d, real);

}
```

4. Pointeri constanti si pointeri la constante.

In definitiile:

```
const int x=10, *px=&x;
```

x este o constanta, în timp ce px este un pointer la o constanta. Aceasta însemna ca x, accesibil si prin px nu este modificabil (operatiile x++ si (*px)++ fiind incorecte, dar modificarea pointerului px este permisa (px++ este corecta).

Un pointer constant (nemodificabil), se defineste prin:

```
int y, * const py=&y;
```

In acest caz, modificarea pointerului (py++) nu este permisa, dar continutul referit de pointer poate fi modificat ((*py)++).

Un pointer constant (nemodificabil) la o constanta se defineste prin:

```
const int c=5, *const pc=&c;
```

In cazul folosirii unor parametri pointeri, pentru a preveni modificarea continutului referit de acestia se prefera definirea lor ca pointeri la constante. De exemplu o functie care compara doua siruri de caractere are prototipul:

```
int strcmp(const char *s, const char *d);
```

. . . . . .
. . . . . .
. . . . . .

Imagini din documentul complet:

Mai multe detalii se gasesc in [pagina documentului](#) din [Biblioteca.RegieLive.ro](#)