



- Exempu



p

 α

X

100

adresa de memorie α

Cod în C:

```
int x = 100;  
int *p = &x;
```

- Orice pointer are un anumit **tip**

- În exemplul de mai sus

- **x** este o variabilă de tip **int** având valoarea 100
- **p** este un pointer de tip **int** având valoarea α

- Un pointer (o adresă de memorie) ocupă în memorie

- 4 octeți, dacă programul rezultat este o aplicație pe 32 de biți (*în întregul curs de Programarea Calculatoarelor considerăm acest caz*)
- 8 octeți, dacă programul rezultat este o aplicație pe 64 de biți



Declararea pointerilor.

Asignarea adresei unei variabile

- Caracterul * precede identificatorul pointer-ului

`tip *identificator`

- Exemplu

`int *p;`

- Adresa unei variabile se poate obține utilizând **operatorul referință &**
- Asignarea adresei variabilei întregi **x** la un pointer **p**:

- Declarații

`int x;`

`int *p;`

- Asignarea adresei

`p=&x;`



- ```
int x, y, *p;
```

- ```
p=&x;  
*p=y;
```

```
p=&y;  
x=*p;
```




- ```
int* p; /* declara pointer-ul, insa acesta nu
 refera o zona de memorie alocata */
```

```
p = 10; / dereferentierea unei zone de memorie
nealocate cauzeaza o eroare serioasa
in momentul executiei programului */
```

- 6



# Pointeri la *void*

- Când un pointer nu trebuie să aibă un anumit tip se utilizează

```
void *identifier;
```

- Utilizarea pointerilor la **void** asigură genericitate implementărilor

- Un pointer la **void** poate primi valoarea unui pointer de orice tip
- Un pointer la **void** nu poate fi dereferențiat

- Exemplu

```
int x;
float y;
void *p;
```

- Atribuirii corecte: **p = &x; p = &y;**
- Pentru dereferențiere trebuie specificat mai întâi tipul (operație de cast) **(tip \*)p** și apoi efectuată dereferențierea

```
p = &y; float z = *((float *)p);
```



- ```
int x;  
void *p;
```

```
p = &x;  
*((int *)p) = 10;
```

Eroare:

*p = 10;





- ```
const tip* identificator=valoare;
tip const* identificator=valoare;
```

- Pointer-ul **identificator** este un pointer la o zonă de memorie care conține o valoare constantă de tipul **tip**
- Valoarea stocată la adresa respectivă nu se va putea schimba

- ```
double z = 4.52;
const double* x = &z;
double y = 3.89;
```

Atribuire imposibilă (valoarea stocată la adresa **x** nu se poate schimba!)

$$*x = y;$$

Atribuire permisă (adresa **x** poate fi schimbată cu adresa lui **y**)

```
x = &y;
```



- ```
const tip* const identifier=valoare;
```

- ```
double z = 4.52;
const double* const x = &z;
double y = 3.89;
```

$$*x = y;$$

```
x = &y;
```



- ```
const tip *parametru_formal
```





- ```
double a[100];
double *p;
p=&a[10];
printf("%p\n", p); // 0028fc38
p++; /* p refera acum elementul a[11].
      Valoarea lui p este incrementata cu
      dimensiunea unui double - 8 octeti*/
printf("%p\n", p); // 0028fc40
```



- Adunarea/scăderea unui întreg la/dintr-un pointer
 - Operațiile $p+n$ și $p-n$ rezultă în incrementarea respectiv decrementarea valorii lui p cu ($n \times$ numărul de octeți) necesari pentru a memora o valoare de tipul lui p
- Exemplu

```
double a[100];
double *p = a+4; /* p este adresa de inceput a
                  tabloului a plus 32 octeti
                  (4 elemente x 8 octeti),
                  p referă pe elementul a[4] */
printf("%p;%p\n", a, p); // 0028fbe0;0028fc00

/* urmatoarele trei instructiuni sunt
   echivalente, x fiind valoarea lui a[4] */

double x = a[4];
double x = *p;
double x = *(a+4);
```



Operații cu pointeri

- Diferența a doi pointeri
 - Dacă doi pointeri **p** și **q** referă elementele de pe pozițiile **i** și **j** dintr-un tablou **a**, adică **p=&a[i]** și **q=&a[j]** atunci **q-p=j-i**
 - Diferența a doi pointeri reprezintă numărul de octeți dintre cele două adrese de memorie împărțit la numărul de octeți pe care este reprezentat tipul de date al pointerilor respectivi
- Exemplu

```
double a[100];  
double *p = a+8; /* p referă elementul a[8] */  
double *q = a+10; /* p referă elementul a[10] */  
int dif = q-p;  
printf("%p;%p;%d\n",p,q,dif); //0028fc20;0028fc30;2
```




Operații cu pointeri

- Compararea a doi pointeri
 - Se poate face cu ajutorul operatorilor `<` `<=` `>` `>=` `==` `!=`
 - Rezultatul este dat de compararea valorilor adreselor de memorie pe care le conțin acei pointeri
- Exemplu

```
double a[100];  
double *p = a+8; /* p referă elementul a[8] */  
double *q = a+10; /* p referă elementul a[10] */  
double *r = &a[8]; /* r referă elementul a[8] */  
printf("%d\n", p>q); // 0  
printf("%d\n", p==r); // 1  
printf("%d\n", q!=r); // 1  
printf("%d\n", p<=q); // 1
```



```
void max_min1(int n, int a[], int *max, int *min)
{
```

```
void max_min2(int n, int *a, int *max, int *min)
{
```

18



Pointeri – exemplul 1

```
int main()
{
    int i, n, maximum, minimum, x[100];
    printf("Dimensiunea tabloului este:");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("x[%d]=", i);
        scanf("%d", &x[i]);
    }

    /* Apelul primei functii */
    max_min1(n, x, &maximum, &minimum);
    printf("max_min1: maximum=%d si minimum=%d\n", maximum, minimum);

    /* Apelul celei de-a doua functii -> utilizand operatii cu pointeri*/
    max_min2(n, x, &maximum, &minimum);
    printf("max_min2: maximum=%d si minimum=%d\n", maximum, minimum);

    return 0;
}
```



20



- ## Atenție!

- 21



Pointeri ca argument și valoare returnată

```
int* f(int* a, int* b) {
    int *c=(*a<*b)?a:b;
    return c;
}

int* g(int* a, int* b) {
    int val=(*a<*b)?*a:*b;
    int *c=&val;
    return c; // adresa unei variabile locale automate (alocate pe stiva)!
}

int main() {
    int x=30;
    int y=60;
    int z=*(f(&x,&y));
    printf("%d\n",z); // 30 -> cu siguranta!
    int t=*(g(&x,&y)); /* este posibil ca zona de memorie unde este alocat
                        "val" sa fie eliberata imediat dupa apelul lui "g" */
    printf("%d\n",t); /* 30 -> numai in cazul in care dereferentierea din
                        instructiunea precedenta nu a esuat! */
    return 0;
}
```