Matricele

- Masive bidimensionale ce permit accesul elementelor pe linii și coloane
- Memoria fiind liniară (unidimensională) pentru a putea fi salvate sunt liniarizate
- Cele alocate static sunt salvate în memorie ca vectori consecutivi ce conțin elementele de pe fiecare linie
- Cele alocate dinamic sunt salvate ca vectori de vectori (un vector ce conține adresele de memorie ale fiecărui vector corespunzător fiecărei linii)

Definire

```
//matrice alocata static de 3 linii si 4 coloane
int matrice[3][4];

//matrice alocata dinamic de 3 linii si 4 coloane
int** m = (int**) malloc(3 * sizeof(int*));
for(int i=0; i<3; i++)
{
     m[i] = (int*) malloc(4 * sizeof(int));
}
```

Dezalocare

```
for(int i=0; i<3; i++)
{
    free(m[i]);
}
free(m);
```

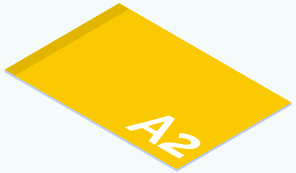
Definire cu și dezalocare cu new și delete

```
int** m = new int*[3];
for(int i=0; i<3; i++)
{
      m[i] = new int[4];
}
//...
for(int i=0; i<3; i++)
{
      delete[] m[i];
}
delete[] m;
```

Legătura dintre pointeri și operatorul []

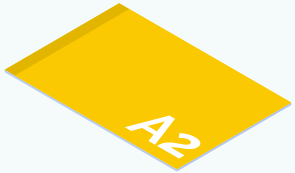
- Operatorul [] se folosește de aritmetica pointerilor pentru a accesa elementele dintr-un masiv
- Așadar:
  - `vector[i] ⇔ *(vector + i)`
  - `matrice[i][j] ⇔ *(*(matrice + i) + j)`

Vectorii de caractere



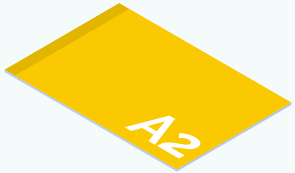
- În limbajul C nu există un tip de dată special definit pentru șiruri de caractere
- Pentru a salva totuși astfel de șiruri se folosește o convenție: se utilizează un vector de caractere (vector de char) ce are drept ultim element '\0' (cod ASCII 0) pentru a ști când vectorul de caractere se sfârșește
- Vectorii de caractere pot fi prelucrați ca orice alt vector sau, profitând de faptul că putem ști când ajungem la ultimul element, prin utilizarea de funcții specifice

Utilizare vectori de caractere



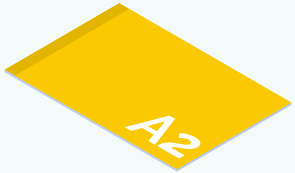

```
char sir[20];
char* sir_dinamic = new char[10];

strlen(sir); //returneaza numarul de caractere al sirului (fara a numara \0)
strcpy(sir, "ceva"); //copiaza sirul din dreapta in variabila din stanga
strcpy_s(sir, 5, "ceva"); //varianta safe
strcmp(sir, sir_dinamic); //compara doua siruri de caractere si returneaza -1, 0 sau 1
strcat(sir, sir_dinamic); //concateneaza sir_dinamic la sfarsitul lui sir
strcat_s(sir, sir_dinamic); //varianta safe
//...
delete[] sir_dinamic;
```

Tipul string



- În limbajul C++ avem tip de dată special definit pentru șiruri de caractere - string
- string este o clasă în C++ (vom descoperi în curând ce este o clasă) așadar prelucrările se fac în mod direct prin operatori sau metode (descoperim și ce este o metodă mai târziu 😃)

Tipul string




```
string sir1;
string sir2 = "ceva";

sir2.length(); //returneaza numarul de caractere al sirului (fara a numara \0)
sir1 = sir2; //copiaza sirul din dreapta in variabila din stanga
(sir1 == sir2) ? 1 : 0; //compararea a doua stringuri
sir1 + sir2; //concatenarea a doua stringuri
```

Transmiterea parametrilor

- În mod implicit parametrii sunt transmiși prin valoare în funcții
- Asta înseamnă că aceștia se copiază în variabile locale ce sunt dezalocate la ieșirea din funcție
- În C++ mai există două modalități adiționale de a transmite parametri:
  - prin adresă (pointer)
  - prin referință (doar în C++, nu și în C)

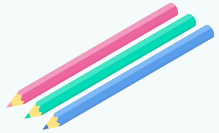
Transmiterea parametrilor prin valoare

```
void suma(int a, int b) {
     a = a + b;
}

int main() {
     int x = 5;
     int y = 3;
     suma(x, y);
     cout << x;
}
```

Transmiterea parametrilor prin adresă

```
void suma(int* a, int b) {
	(*a) = (*a) + b;
}

int main() {
	int x = 5;
	int y = 3;
	suma(&x, y);
	cout << x;
}
```

Transmiterea parametrilor prin referință



```
void suma(int& a, int b) {
	a = a + b;
}

int main() {
	int x = 5;
	int y = 3;
	suma(x, y);
	cout << x;
}
```

Pointeri la funcții

- Atât în C, cât și în C++ putem utiliza pointeri care salvează adresa de memorie a unor funcții
- Aceștia implementează o formă incipientă de polimorfism (un pointer la funcție poate face diferite tipuri de prelucrări prin schimbarea adresei către care pointează)

```
tip_returnat (* nume_pointer) (parametri);
```

Exemplu

```
long (*pf)(int, int); //definire pointer la o functie ce returneaza long
//si are doi parametri de tip int

long suma(int a, int b) { return a + b; }
long diferenta(int a, int b} { return a-b; }
pf = suma;
pf(5, 3); //returneaza 8
pf = diferenta;
pf(5, 3); //returneaza 2
```

Directive de preprocesare

```
#define NULL 0
#define BEGIN {
#define END }
//...
#undef NULL
//...
#if, #elif, #else, #endif, s.a.
```

Curs 03

CLASE

Definire, atribute, metode

Enumerările

- Construcții ce permit crearea unui tip de dată ce acceptă valori doar dintr-un interval finit dat
- Exemple: culori, zilele săptămânii, lunile anului, etc
- La compilare sunt transformate în constante întregi
- Ajută la evitarea unor erori de execuție și dau o mai mare lizibilitate codului sursă

Exemple

```
//definire
enum culoare { rosu, galben, albastru };
enum ziLibera { luni = 1, miercuri = 3, vineri = 5 };

//utilizare
culoare c = culoare::rosu;
ziLibera zi = ziLibera::miercuri;
zi = (ziLibera)5;
```

Uniunea

- Reprezintă un tip de dată ce poate salva o singură valoare la un moment dat dintre mai multe valori posibile
- Uniunea ocupă un spațiu în memorie echivalent cu spațiul ocupat de cel mai mare dintre tipurile definite în interiorul ei
- Zona respectivă de memorie este interpretată apoi în funcție de tipul cerut dintre cele definite

Exemplu

```
//definire
union Eticheta
{
  short v1;
  int v2;
  long long v3;
};

//utilizare
 |union| Eticheta eticheta;
 eticheta.v3 = 9999999;
 printf("%lld\n", eticheta.v3); printf("%d\n", eticheta.v1); printf("%d\n", eticheta.v2);
 eticheta.v2 = 7;
 printf("%lld\n", eticheta.v3); printf("%d\n", eticheta.v1); printf("%d\n", eticheta.v2);
```

Structurile

- Construcții ce permit crearea unui tip de dată complex ce grupează mai multe caracteristici ale unei entități din lumea reală
- Implementează o formă incipientă de încapsulare
- Permit prelucrarea unitară a unei entități
- În C permit doar declararea de atribute (câmpuri, proprietăți) aferente acelei entități (mai multe detalii despre C++ imediat)

Exemplu

```
//definire
|typedef| struct Carte
{
	int numarPagini;
	float pret;
	char* denumire;
	char* autor;
};
//utilizare
|struct| Carte carte; carte.numarPagini = 200; carte.pret = 99.99;
carte.denumire = nullptr; printf("%s", carte.denumire);
```