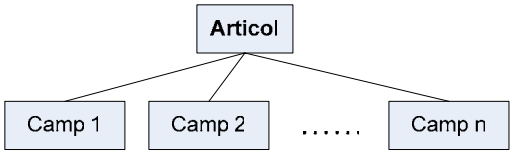
Articole

Articolele sunt structuri de date neomogene si continue. Ele au un numar fix de elemente numite campuri, iar accesul la acestea se face direct.

Utilitatea articolelor apare in manipularea datelor complexe (care presupun stocarea unui numar de caracteristici pentru fiecare entitate) si in crearea structurilor de date evoluate.

Reprezentarea grafica a unui articol se poate face sub forma:



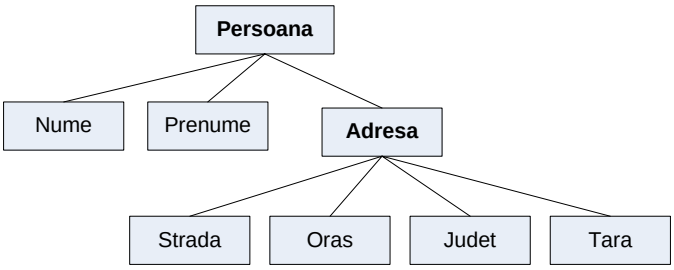
Campurile care compun un articol pot avea orice tip de date, inclusiv articol. In acest fel se pot defini structuri complexe de date.

Exemplu:

Persoana (articol):

- Nume
- Prenume
- Adresa (articol)
  - Strada
  - Oras
  - Judet
  - Tara

Reprezentarea acestei structuri este:



Dimensiunea unui articol se calculeaza recusiv ca suma dimensiunilor campurilor componente.

1. Declarare si initializare

Sintaxa folosita pentru definirea unui articol in C++ este:

struct nume_structura { declaratii campuri } lista_variabile;

unde:

- **nume_structura**: numele tipului de date definit;

- **declaratii campuri**: declaratiile de campuri (similara cu declaratiile de variabile;
- **lista_variabile**: lista de variabile de tipul definit.

nume_structura si **lista_variabile** pot lipsi, dar nu simultan. In cazul in care lipseste numele structurii nu se vor putea declara ulterior variabile si nu se vor putea transmite ca parametri. In cazul in care nu este specificata o lista de variabile, acestea se pot declara ulterior folosind sintaxa ***nume_structura lista_variabile***.

Exemple de articole:

```
// declarare structura 'Adresa'
struct Adresa
{
	// declaratii campuri
	char strada[100], oras[50], tara[50];
	int cod_judet;
};

// declarare structura 'Student'
// si variabile de tip student
struct Student
{
	// campuri simple
	char nume[50], prenume[50];

	// camp de tip structura
	Adresa adresa;
} student1, student2;

// declarare variabile de tip adresa
Adresa o_adresa, alta_adresa;
```

Definirea unui articol este de fapt definirea unui nou tip de date in cadrul limbajului. Aceasta nu presupune alocare de memorie, dar permite declararea ulterioara de variabile din noul tip. Compilatorul va aloca memorie doar pentru variabilele declarate pe baza definitiei articolului. Pentru determinarea dimensiunii efective de memorie utilizate se recomanda folosirea operatorului ***sizeof,*** deoarece ea poate fi diferita de suma dimensiunilor campurilor componente (datorita restrictiilor de aliniere).

Variabilele de tip articol se pot initializa la declarare folosind sintaxa:

nume_structura variabila = {lista de initializare};

Exemplu:

```
	// declarare si initializare
	Student s = {
		// valori campuri simple
		"Popescu", "Ion",
		// valori campuri de tip structura
		{"Str. Zambilelor 2", "Buhusi", "Romania", 34} };
```

Limbajul C++ permite definirea unui tip special de campuri numite campuri de tip bit. Aceste campuri permit o folosire mai eficenta a memoriei in cazul in care o

caracteristica are un set de valori foarte mic (ex: sex, an de studii). Aceste campuri pot fi combinate cu campurile normale in cadrul unei structuri. Sintaxa folosita este:

tip nume_camp:nr_biti;

Exemplu:

```
struct Elev
{
	// clasa -> 1-12
	unsigned short clasa : 4;
	// 0 - M, 1 - F
	unsigned short sex : 1;

	char nume[100];
};
```

Campurilor de tip bit se comporta ca si cele normale, dar au o serie de particularitati:

- tipul campului trebuie sa fie de tip intreg
- un camp nu trebuie sa taie limita unui cuvant
- lungimea unui camp nu poate depasi lungimea unui cuvant
- numele campului poate lipsi, caz in care nu poate fi accesat, dar este folosit pentru aliniere
- nu se poate extrage adresa unui astfel de camp

2. Operatii cu articole

Accesul la membrii structurii se face folosind sintaxa ***variabila.camp***.

Exemple:

```
	// extrage prima litera din nume
	char a = s.nume[0];

	// extrage codul din structura imbricata
	int cod = s.adresa.cod_judet;

	// modifica codul judetului
	s.adresa.cod_judet = 27;
```

Operatiile premise asupra datelor de tip articol sunt:

a) extragerea adresei folosind operatorul &:

```
	Student *pStudent = &s;
```

Accesarea campurilor prin intermediul pointerului se poate face folosind operatorul de indirectare * sau operatorul ->: `(*pStudent).nume sau pStudent->nume;`

b) atribuirea la nivel global folosind operatorul = :

```
	Student s1;
	s1 = s;
```

Copierea structurilor se face bit cu bit. In cazul in care structura contine pointeri, zonele de memorie referite trebuie copiate manual de catre programator. Singura exceptie o constituie masivele, care sunt copiate automat de compilator.

c) transmiterea ca argument in functie si returnarea ca rezultat:

```
Student PrelucrareStudent(Student s)
{
   Student tmp;
   ....
   return tmp;
}
```

Structurile de date, spre deosebire de massive, sunt trimise prin valoare. In cazul in care structura contine un masiv, acesta va fi copiat in intregime la momentul apelului.

3. Campuri de dimensiune variabila

Limbajul C++ permite definirea de campuri de lungime variabila numite uniuni. Uniunea corespunde unei zone de memorie care la momente de timp diferite poate contine elemente de tipuri si lungimi diferite.

Ca si structurile, uniunile sunt formate dintr-un ansablu de campuri. In cazul uniunilor, campurile sunt alocate in acelasi spatiu de memorie (ele sunt mutual exclusive). Din acest motiv, uniunea se comporta ca o structura in care toate elementele sunt aliniate la aceeasi adresa. Lungimea unei uniuni este data de lungimea celui mai lung camp.

Sintaxa de declarare este:

union nume {lista_campuri};

Exemplu de utilizare:

```
// declarare uniune
// (poate contine un numar intreg
// sau un numar real)
union numar
{
      int intreg;
      double real;
};


// declarare variabila
numar n;

// folosire ca intreg
n.intreg = 7;
int i = n.intreg;

// folosire ca real
n.real = 5.6;
double d = n.real;
```

Interpretarea continutului zonei cade in sarcina programatorului. In general, pentru a putea retine usor tipul informatiilor prezente la un moment dat intr-o uniune, aceasta se include impreuna cu o variabila care mentine tipul stocat intr-o structura.

De exemplu, pentru a mentine date despre elevi si studenti se poate crea o stuctura de forma:

```
// declarare uniune
union DatePersoana
{
	Student student;
	Elev elev;
};

// includere uniune in articol
struct Persoana
{
	// tipul persoanei: 0 - Student, 1 - Elev;
	int tip;
	DatePersoana date;
};
```

La momentul prelucrarilor se va verifica intai tipul stocat, iar dupa aceea se vor efectua prelucrarile.

De exemplu, functia de afisare poate avea forma:

```
void AfisarePersoana(Persoana pers)
{
	// determinare tip
	if (pers.tip == 0)
	{
		// afisare student
		cout << pers.date.student.nume << " "
			<< pers.date.student.prenume << endl;
	}
	else
	{
		// afisare elev
		cout << pers.date.elev.nume << " "
			<< pers.date.elev.clasa << endl;
	}
}
```

4. Vectori de articole

Structurile pot fi grupate in masive, la fel ca oricare tip de date. De asemenea, structurile pot vectori.

Exemple:

a) vector in articol

```
struct Student
{
```

```
	int cod;
	int nr_note;
	// vector in structura
	int note[10];
};
```

b) vector de articole

```
// declarare si initializare
// vector de studenti
Student grupa[] =
{
	{1, 3, {7, 10, 10}},
	{2, 3, {10, 8, 9}},
	{3, 2, {10, 10}}
};
```

5. Probleme

1. Pentru o factura se dau urmatoarele date: informatii despre beneficiar si cumparator si lista de produse (cantitate si pret unitar). Sa se defineasca o structura de date potrivita pentru stocarea informatiilor referitoare la o factura si sa se scrie o functie care calculeaza celelalte informatii prezente pe factura (valoare totala si TVA pentru fiecare produs si per total).
2. Scrieti programul care realizeaza initializarea si referirea unei structuri de date de tip uniune. Explicati rezultatele programului in urma executiei.
3. Despre fiecare student dintr-o grupa se cunosc numele acestuia, notele obtinute la seminar, numarul de prezente la seminar. Sa se memoreze intr-o structura de date corespunzatoare toate aceste informatii si sa se scrie functiile si procedurile pentru obtinerea urmatoarelor situatii::
    a. afisarea studentilor in ordine alfabetica
    b. afisarea informatiilor despre un anumit student
    c. inscrierea unui nou student in grupa
    d. stergerea unui student din grupa
4. O societate comerciala cu profil de productie mentine informatii de gestiune referitoare la materiile prime folosite si la produsele finite realizate. Pentru materiile prime se retin: codul materialului, grupa de produse (max. 16 grupe), subgrupa de produse (max 16 subgrupe), daca este produs local sau importat, daca necesita conditii speciale de depozitare sau nu, cantitatea disponibila in stoc si pretul unitar. Datele asociate unui produs sunt: codul produsului, cantitatea existenta in stoc, pretul unitar si consumurile specifice (max 10). Se cere:
    a. Sa se defineasca o structura unitara si eficienta pentru tinerea gestiunii.
    b. Sa se scrie functii pentru:
        i. Afisarea listei de gestiune (tip, cod, cantitate, valoare totala);
        ii. Determinarea valorii stocurilor de materii prime, de produse finite si valoarea totala a elementelor din gestiune;
        iii. Determinarea valorii totale si pe subgrupe a stocurilor pentru o grupa de materii prime;
        iv. Determinarea profitului obtinut in cazul vanzarii intregului stoc de produse finite;
        v. Determinarea pentru un produs dat a numarului maxim de unitati ce pot fi produse pe baza stocurilor existente.

Rezolvare:

```
struct mat_pr
{
	int cod;
	short grupa:4;
	short subgrupa:4;
	short local:1;
	short cond_spec:1;
	int cant, pret;
};

struct consum
{
	int cod_mat, cant;
};

struct prod_fin
{
	int cod;
	int cant, pret;
	int nr_mat;
	consum consumuri[10];
};

union date_elem_gest
{
	prod_fin prod;
	mat_pr mat;
};

struct elem_gest
{
	short tip;
	date_elem_gest date;
};


void AfisareLista(elem_gest gest[], int n)
{
	for(int i = 0; i < n; i++)
	{
		if (gest[i].tip == 0) // daca e material
			cout << "mat  " << gest[i].date.mat.cod
				<< " " << gest[i].date.mat.cant
				<< " " << gest[i].date.mat.cant * gest[i].date.mat.pret
<< endl;
		else
			cout << "prod " << gest[i].date.prod.cod
				<< " " << gest[i].date.prod.cant
				<< " " << gest[i].date.prod.cant *
gest[i].date.prod.pret << endl;

	}
}

void ValoareStocuri(elem_gest gest[], int n)
{
	int totalMat = 0, totalProd = 0;

	for(int i = 0; i < n; i++)
		if (gest[i].tip == 0) // daca e material
			totalMat += gest[i].date.mat.cant * gest[i].date.mat.pret;
		else
			totalProd += gest[i].date.prod.cant * gest[i].date.prod.pret;

	cout << "Total materii prime: " << totalMat << endl;
	cout << "Total produse: " << totalProd << endl;
	cout << "Total general: " << totalMat + totalProd << endl;
}

void ValoareStocuri(elem_gest gest[], int n, int grupa)
{
	int i, totalSubGr[16], total = 0;
	for (i = 0; i < 16; i++)
		totalSubGr[i] = 0;
```

```
	for(i = 0; i < n; i++)
		if (gest[i].tip == 0 && gest[i].date.mat.grupa == grupa)
		{
			totalSubGr[gest[i].date.mat.subgrupa] += gest[i].date.mat.cant *
gest[i].date.mat.pret;
			total += gest[i].date.mat.cant * gest[i].date.mat.pret;
		}
	for (i = 0; i < 16; i++)
		cout << "Total subgrupa " << i << ": " << totalSubGr[i] << endl;
	cout << "Total grupa: " << total;
}

void DeterminareProfit(elem_gest gest[], int n)
{
	int profitTotal = 0;

	for(int i = 0; i < n; i++)
		// pentru fiecare produs
		if (gest[i].tip == 1)
		{
			// determinare cost produs
			int cost = 0;
			prod_fin p = gest[i].date.prod;
			for (int j = 0; j < p.nr_mat; j++)
				for(int k = 0; k < n; k++)
					if (gest[k].tip == 0 && gest[k].date.mat.cod ==
p.consumuri[j].cod_mat)
						cost += p.consumuri[j].cant *
gest[k].date.mat.pret;

			// adaugare profit pentru produs la profitul total
			profitTotal += p.cant * (p.pret - cost);
		}

	cout << "Profit total:" << profitTotal << endl;
}

void NumarMaximUnitati(elem_gest gest[], int n, int codProd)
{
	// cautam produsul
	prod_fin p;
	for(int i = 0; i < n; i++)
		// pentru fiecare produs
		if (gest[i].tip == 1 && gest[i].date.prod.cod == codProd)
			p = gest[i].date.prod;

	int nrMax = -1;

	// pentru fiecare materie prima necesara
	for (int j = 0; j < p.nr_mat; j++)
		for(int k = 0; k < n; k++)
			if (gest[k].tip == 0 && gest[k].date.mat.cod ==
p.consumuri[j].cod_mat)
			{
				int prodRealizabile = gest[k].date.mat.cant /
p.consumuri[j].cant;

				// daca nu e initializat sau daca numarul estimat
				// pana acum este prea mare
				if (nrMax == -1 || nrMax > prodRealizabile)
					nrMax = prodRealizabile;
			}

	cout << "Numar maxim de produse realizabile: " << nrMax << endl;
}
```