ХИМИКОТЕХНОЛОГИЧЕН И МЕТАЛУРГИЧЕН УНИВЕРСИТЕТ – СОФИЯ

ИНФОРМАТИКА

част първа

Основи на програмирането на C++

лектор: гл. ас. д-р Стефан М. Панов

Катедра “Информатика”

Лекция 7

Масиви в C++

Масив (array) - съвкупност от подредени променливи от един и същи тип и с общо име. Променливите, които принадлежат на масива, се наричат още елементи. Масивите могат да бъдат едномерни и многомерни.

В практиката най-често се използват едномерни и двумерни масиви, по-рядко тримерни или с по-голяма размерност.

Едномерни масиви

Едномерен масив се определя така:

тип име_на_масив[размер];

където **размер** задава броя на елементите на масива. Може да бъде само неотрицателен целочислен константен израз.

тип определя типа на елементите на масива.

Квадратните скоби след името указват на компилатора, че това е масив.

Типът може да бъде всеки един от базовите типове. Съществуват и 4 други типа на масив (указателен-тип, структурен-тип, union-тип и enum-тип), но те ще бъдат разгледани по-нататък.

Достъпът до елементите на масива става чрез неговото име, последвано от номера (индекса) на съответния елемент в квадратни скоби, т.е.

име_на_масив[индекс]

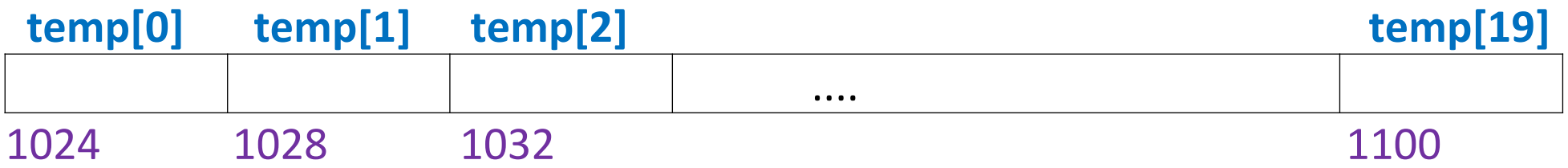
където индекс може да бъде всеки целочислен израз.

Забележете, че е позволено изразът да има и отрицателна стойност, но това е груба грешка.

Елементите на масива се номерират от 0 нагоре, т.е. първият елемент има индекс 0, вторият индекс 1 и т.н. до последният, имащ индекс **размер – 1**.

Пример: **int temp[20];** обявява се масив от целочислен тип на име **temp** съдържащ 20 елемента. Първият е **temp[0]** а последният **temp [19]**.

Когато компилаторът срещне определение на масив, той разполага **последователно** неговите елементи в паметта на компютъра. За примера:



Типът **int** има размерност 4 байта, т.е. за нашият пример ще са необходими 80 байта в паметта на компютъра. Ако адресът на първият байт от заделената памет е примерно 1024 това означава, че адресът на първия елемент **temp[0]** е 1024, на втория елемент е 1028 и т.н. Лесно се вижда че:

Брой_заделени_байтове = sizeof(тип) * размер_на_масива

Елементите на масива могат да се използват навсякъде, където обикновени променливи от същия тип могат да бъдат използвани. Тоест **име_на_масив[индекс]** се използва както име на обикновена променлива. Има връзка между вектор в математиката и едномерен масив.

В следващия пример се обявява масив от **N** елемента, където **N** е именована константа и на всеки елемент се присвоява случайно число. После се намира и извежда на екран минималният елемент (елементът имащ минимална стойност).

```
#include <iostream> // nashata 28-a programa
#include <cstdlib>  // namirane na min element na masiv ot N elementa
#include <time.h>
using namespace std;
const int N = 10; // razmer na masiva
int main() {
	int i, min_value;
	int m[N]; // masiv ot N elementa
	srand(time(NULL)); // srand() i time() sa razgledani predi
	for(i=0; i<N; i++) {
		m[i] = rand(); // generirame N celi polojitelni chisla
	}
```

```
    min_value = m[0]; // v nachaloto vremennia minimum e pyrviat element
    for (i=1; i<N; i++){// namirame minimalnia element taka:
        if(min_value > m[i]) { // ako tekushtiqt element e po-malyk ot vremennia
            min_value = m[i];// minimum toi stava minimum
        }
    }
    cout << "Elementite na masiva sa:\n\n";
    for(i=0; i<N; i++) {
        cout << m[i] << " ";
    }
    cout << "\n\nMinimalnata stoinost e  " << min_value << "\n";
    return 0;
}
```

В началото минималният елемент е първият. Временния минимум се сравнява последователно със всички останали (забележете, че вторият for се

инициализира с единица) елементи от масива. Ако текущият елемент е по-малък от временния минимум неговата стойност става минимална.

Примерът е добра илюстрация на ползите от именованите константи. Програмата може да бъде написана и без реда `const int N = 10;` като навсякъде вместо `N` се постави числото 10. Но евентуална промяна на размера на масива би изисквала да сменим 10 с новата стойност отново навсякъде, където 10 се среща. Това първо е трудоемко и второ съществува опасността да забравим да го сменим накъде, особено в по-големи програми. Последното води до трудно-откриваеми грешки! При използването на константа промяната е само една, примерно `const int N = 12;`

Внимание: Езикът C не следи дали индексът е в границите на масива. Ако индексът излезе извън границите на масива, поведението е недефинирано.

Употребата на масив със сгрешен индекс, особено в инструкция за присвояване може да промени стойностите в клетки от паметта, заделени за други променливи от вашата програма или даже в клетки извън вашата програма. Това води до тежки последствия без каквито и да е забележки от

страна на компилатора и без издаване на съобщения за грешки по време на работата на програмата. Това означава, че цялата отговорност за съблюдаване на границите на масивите пада изцяло върху програмистите!

Накратко: Внимавайте при индексиране на масива да не излезете извън границите на масива. Винаги проверявайте дали индексът е валиден.

Инициализация на масиви

Масивите могат да се инициализират по време на дефинирането си. Общата форма на инициализация на едномерен масив изглежда така:

тип име_на_масив [размер] = {инициализатор1, … , инициализаторN};

Инициализаторите могат да бъдат само константи. Броят на инициализаторите не трябва да превишава броя на елементите на масива, зададен с размер. Ако броят на инициализаторите е по-малък, елементите, за които не достигат инициализатори, се инициализират с нула. Пример:

```
#include <iostream> // nashata 29-a programa
using namespace std; // inicializaciq na masiv ot N elementa
const int MONTHS = 12; // razmer na masiva
int main() {
	int days[MONTHS] = {31,28,31,30,31,30,31,31,30,31,30,31};
	int index;
	for (index = 0; index < MONTHS ; index++) {
		cout << "Mesec " << index + 1 << " ima " << days[index] << " dena.\n";
	}
	return 0;
}
```

Когато едномерен масив се инициализира по време на дефинирането си, размерът на масива може да се пропусне. В тоя случай компилаторът определя броя на елементите на масива от броя на инициализаторите.

Да си припомним, че в лекция 4 се запознахме с низовите константи. **По същество низовата константа (низът) е едномерен масив от символи.** Низовите константи могат да се използват за инициализиране на символни масиви:

char име_на_масив[размер] = "tova e niz";// razmer e pone 11 simvola

или

char име-масив[ ] = "tova e niz";

Обърнете внимание, че размерът на символния масив трябва да е с единица по-голям от броя на символите в низа заради нулевия символ.

В C++ нулевият символ се използва за обозначаване на край на низ и се означава като '\0'. Компилаторът автоматично добавя нулев символ след последния символ.

t	o	v	a		e		n	i	z	'\0'			

Символният масив може да се инициализира и символ по символ, но трябва ние да добавим и нулевия символ накрая, за да бъде низ. Ето кратка програма за двата вида инициализация:

```
#include <iostream> // nashata 30-a programa
using namespace std; // inicializacia na simvolen
const int N = 11; // masiv ot N elementa po dva nachina
int main() { // vajno e teksta da ne nadvishava razmera na masiva -1
	char niz1[N] = {'T','o','v','a',' ','e',' ','n','i','z','\0'};
	char niz2[N] = "Tova e niz";
	cout << niz1 << "\n";
	cout << niz2 << "\n";
	return 0;
}
```

Обърнете внимание на конструкцията cout << niz1. Единствено в случай когато масивът съдържа символен низ може да се посочва само името му, за да се разпечата целия низ. (Ако масивът съдържа числа ще се отпечата

адресът на първия елемент на масива!) Накратко: **масив, съдържащ низ може да се използва навсякъде, където е допустимо използването на низова константа**.

Масивите не могат да се копират направо, т.е. ако имаме масиви **w** и **v** с еднакъв брой елементи, конструкцията v = w е невалидна. Ако искаме да копираме един масив в друг, ще трябва да го направим елемент по елемент.

Прочитане на низ (ред) от клавиатура

Един възможен вариант е да използваме инструкция cin. Но следващата програма работи изненадващо за начинаещи програмисти:

```
#include <iostream> // nashata 31-a programa
using namespace std; // chetene na niz ot
const int N = 80; // klaviatura s instrukcia cin
int main() { // vajno e teksta da ne nadvishava razmera na masiva -1
    char niz[N];
    cout << "Vavedete niz: ";
```

```
cin >>   niz ;
cout << "Eto kakvo vavedohte: " <<   niz ;
return 0;}
```

Ако въведем текст "Haide da vidim" на екрана ще се покаже само "Haide". Работата е там, че операторът ">>" прекратява въвеждането когато срещне един от следните 3 символа: **интервал, табулация или символа за нов ред**.

За решение на възникналия проблем се използва готовата функция **gets()**. Тя се извиква така:

gets(име_на_масив);

Масивът се задава само с името си, без индекси. **Функцията gets() чете въвежданите от потребителя символи дотогава, докогато не се натисне клавиша <Enter>.** За извикване на функция **gets**() е необходимо в програмата да се включи заглавният файл **<cstdio>** (или <stdio.h>). Коментираме реда с **cin** и добавяме **gets(niz) ;** Сега въведеният от клавиатура текст се чете до края.

Внимание: Нито оператор ">>", нито функцията **gets()** проверяват за нарушение на границите на масива. Т.е., ако потребителят въведе ред, чиято дължина превишава размера на масива са възможни неприятностите заради промяна на чужда памет, за които се спомена по-нагоре.

Двумерни масиви

Двумерният масив е масив, чиито елементи са едномерни масиви. Двумерен масив се определя така:

тип име_на_масив[размер1][размер2];

където

размер1 определя броя на елементите на двумерния масив, т.е. броя на

едномерните масиви. Може да бъде само целочислен положителен константен израз. **размер2** определя броя на елементите в едномерните масиви. Може да бъде само целочислен положителен константен израз.

тип определя типа на елементите на едномерните масиви.

Елементите на двумерните масиви се достигат също чрез индексиране.

име_на_масив [индекс1] [индекс2]

където **индекс1/индекс2** може да бъде всеки целочислен израз.

Двумерният масив може да се представи графично като таблица (матрица), в която **размер1** определя броя на редовете, а **размер2** определя броя на колоните. С **индекс1** се избира реда (т.е. едномерния масив), а с **индекс2** се избира конкретния елемент от тоя масив (т.е. колоната).

Двумерните масиви също могат да се инициализират при дефинирането си.

Казаното за едномерни масиви: „Ако броят на инициализаторите е по-малък, то елементите, за които не достигат инициализатори се инициализират с нула.“ е в сила и за двумерни и многомерни масиви .

Както и при обикновените променливи, глобалните масиви се инициализират в началото на изпълнението на програмата, а локалните — при всяко викане на функцията, в която те се съдържат.

В показаната програма_32 се отпечатва квадратът на число от 1 до 10. Всички стойности на масива са инициализирани, като стойностите за всеки от десетте едномерни масива също са поставени в къдрави скоби. Същите тия вътрешни скоби могат и да се пропуснат, това е разрешено от синтаксиса на езика C++. Лесно се съобразява, че не е нужно да търсим стойността, въведена от клавиатура в променливата **i**, а можем веднага да отпечатаме съответния квадрат. С цикъла **do-while** пък подсигуряваме, че въведеното число ще бъде в желания от нас интервал. (Забележка: В посочения пример не е подсигурена защита в случай на нарочно или неумишлено въвеждане на друг текст вместо цяло число. Но и за това има решение, както ще видим по-късно.)

Когато едномерен (многомерен) масив се инициализира по време на дефинирането си, размер (размер1) може да се пропусне, компилаторът определя броя на елементите на масива от броя на инициализаторите. Т.е за нашия пример е допустим и следният ред int kvadrati[][2] = { Това е удобно, когато плануваме да променяме броя на редовете на масива.

```
#include <iostream> // nashata 32-a programa
using namespace std; // inicializacia na dvimeren masiv
int main() {
    int kvadrati[10][2] = {
    {1, 1}, {2, 4}, // ot nas zavisi na kolko reda shte
    {3, 9}, {4, 16}, // zapishem inicializaciata
    {5, 25}, {6, 36},
    {7, 49}, {8, 64},
    {9, 81}, {10, 100} };
    int i;
    do { // podsiguriavame, che shte vavedem chislo v jelania interval
```

```
        cout << "vavedete cialo chislo ot 1 do 10: ";
        cin >> i;
    } while (i < 1 || i > 10);
    cout << i << " na kvadrat e ravno na " << kvadrati[i-1][1] << "\n";
    return 0;
}
```

Въвеждане/извеждане на елементите на едномерен масив от клавиатура/на екран

Най-важното е да подсигурим правилни стойности на индекса за да не излиза извън границите на масива. Да разгледаме следния пример:
Да се намери стандартното отклонение на мостра от данни по формулата:

$$\sigma = \sqrt{\frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{x})^2} \qquad \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

където σ — стандартното отклонение на мострата (има разнобой в терминологията); σ^2 — дисперсията; x_i — ***i***- тия елемент на данните; $\bar{X}$ — средно аритметическо на данните; n — брой на данните. **Дисперсия на случайна величина - мярка за разпределението на стойностите на случайна величина относно нейното математическото очакване**.

```
#include <iostream> // nashata 33-ta programa
#include <cmath> //namirane na standartnoto otklonenie
using namespace std;//  na stoinostite na sluchaina velichina
const int N = 9; // broi na stojnostite
int main() {
	float x[N];
	int i; // za upravlenie na ciklite v programata
	float sigma; // standartnoto otklonenie
	float xsr = 0; // sredno aritmetichno na x
	float xkv = 0; // za sumata na kvadratite
	for (i=0; i < N;i++){ // vavejdane na ednomeren masiv
		cout << "x["<<i<<"]="; cin >> x[i];
```

```
    }
    for (i=0; i < N;i++){ // namirane na srednata soinost = matemat. ochakvane
        xsr = xsr + x[i];// po-dobre xsr += x[i];
    }
    xsr /= N; // po-dobre e ot xsr = xsr/N;
    for (i=0; i < N;i++){ //
        xkv += (x[i]-xsr)*(x[i]-xsr);
    }
    sigma = sqrt(xkv/(N-1));
   cout << "\n\n Za masiv ot stoinosti:";
    for (i=0; i < N;i++){ // izvejdane na stoinostite ednomeren masiv na edin red
        cout << x[i] << " ";
    }
    cout <<"\n\n standartnoto otklonenie e " << sigma
    << ", a srednata stoinost e " << xsr << "\n";
    return 0;
} // krai na programa_33
```

Да направим разбор на предложеното решение: При избор на променливите е по-добре типът да бъде float или double когато няма конкретни изисквания, че данните трябва да бъдат цели числа. За намирането на средното аритметично е нужна само една променлива, в която първо да намерим сумата на елементите на масива, а после и самата средна стойност. **Винаги, когато ще намираме сума в някаква променлива тя трябва да се инициализира с нула**.

Въпреки, че в дадените от теорията формули индексът се изменя от 1 до n, елементите на масива започват от x[0], а последният елемент е x[N-1]. Но важното е, че отново общият брой стойности е N. След тая уговорка са ясни елементите на първият цикъл for. Въвеждането на стойност за поредния елемент на масива става с инструкция cin. Въпреки, че инструкцията cout преди cin не е задължителна е много по-удобно да знаем кой елемент въвеждаме, особено при големи масиви. За изписването на текущият елемент двете постоянни неща са дадени в низове, а индексът се задава чрез променливата i.

Препоръка: **Винаги, когато имаме въвеждане на стойност в променлива е добре да изпишем преди това с cout подходящ текст, който да улесни въвеждането.** При намиране на средната стойност е казано, че инструкцията xsr = xsr + x[i]; може да се замести с xsr += x[i] Това е така, защото в C++ са предвидени така наречените:

Съставни оператори за присвояване (СОП),

в които е обединено присвояване с още една операция. Обобщено, при следният формат на инструкция за присвояване с бинарни оператори: **променлива = променлива op израз;** новата форма на запис изглежда така: **променлива op = израз;**

Тук **op** означава конкретен аритметически или битов оператор, обединен с оператор за присвояване **=**. Освен, че са удобни СОП позволяват на компилатора да генерира по-ефективен код. Общият брой СОП е 10, виж табл. 7.1 Втората петица оператори (битови) ще бъдат разгледани по-късно.

+=	-=	*=	/=	%=	>>=	<<=	&=	^=	|=

Таблица 7.1 - съставни оператори за присвояване

Последният for в програмата печати масива. Алтернативният негов cout е

cout << "x["<< i <<"]="<< x[i] << "\n";// po edin element na red

Многомерни масиви

В C++, освен двумерни, могат да се определят и масиви с три и повече измерения. Ето как се обявява многомерен масив:

тип име_на_масив [размер1] [размер2] ...[размерN] ;

Например, с помощта на следващата дефиниция се създава тримерен целочислен масив размер 4x10x3.

int multidim[4][10][3];

Многомерните масиви се използват рядко защото заемат много памет. Ако приложим всички разсъждения за едномерни и двумерни масиви, описани по-рано, сами може да се разберем как работи един многомерен масив.

Въвеждане (извеждане) на елементите на двумерен масив от клавиатура (на екран)

Ще разгледаме за демонстрация програма за събиране на две еднотипни (с еднакъв размер) матрици.

```
#include <iostream> // nashata 34-ta programa
#include <cmath> //sybirane na ednotipni matrici
using namespace std;
const int M = 2; // broi na redovete za trite matrici
const int N = 3; // broi na kolonite za trite matrici
int main() {
	float a[M][N];// pyrvata matrica
	float b[M][N];// vtorata matrica
	float c[M][N];// rezultatnata matrica
	int i,k;// za upravlenie na ciklite v programata
	cout <<"Vavedete stoinosti za 2 matrici s razmeri "<<M << " na "<< N << "\n\
n";
```

```
    for (i=0; i < M;i++){ // vavejdaneto na dvumeren masiv
        for (k=0; k < N;k++){  // stava s dva cikyla
            cout << "a["<<i<<"]["<<k<<"]=";
            cin >> a[i][k];
        }
    } cout << "\n";// za preglednost
    for (i=0; i < M;i++){ // vavejdane na dvumeren masiv
        for (k=0; k < N;k++){
            cout << "b["<<i<<"]["<<k<<"]=";
            cin >> b[i][k];
        }
    }
    for (i=0; i < M;i++){ // sybirane na matricite
        for (k=0; k < N;k++){ // sybirat se syotwetnite elementi
            c[i][k] = a[i][k] + b[i][k];
        }
}
```

```
    cout << "\n\nRezultatnata matrica C sled sybiraneto: \n\n";
    for (i=0; i < M;i++){ // izvejdane na dvumeren masiv
        for (k=0; k < N;k++){
            cout <<c[i][k]<< "\t";// "\t" za pravilna podredba
        }
        cout << "\n";// za poredovo otpechatvane
    }
    return 0;
} // krai na programa_34
```

Кратък коментар на програмата:

За въвеждане или извеждане на двумерен (N-мерен) масив са необходими два (N) вложени цикъла, по един за всяко измерение на масива.

Въвеждането и извеждането на масивите в задачата е по редове. Използването на табулация при разпечатването на резултатната матрица улеснява подредбата при различен брой цифри на стойностите на елементите. В следващите лекции ще видим, че има и други възможности.