6

Съставни типове данни. Масив. Символен низ

6.1 Структура от данни масив

Под структура от данни се разбира **организирана информация, която може да бъде описана, създадена и обработена с помощта на програма.**

За да се определи една структура от данни е необходимо да се направи:

- **логическо описание на структурата**, което я описва на базата на декомпозицията ѝ на по-прости структури, а също на декомпозиция на операциите над структурата на по-прости операции.

- **физическо представяне на структурата**, което дава методи за представяне на структурата в паметта на компютъра.

В предходните глави разгледахме структурите числа и символи. За всяка от тях в езика C++ са дадени съответни типове данни, които ги реализират. Тъй като елементите на тези структури се състоят от една компонента, те се наричат **прости**, или **скаларни**.

Структури от данни, компонентите на които са редици от елементи, се наричат **съставни**.

Структури от данни, за които операциите включване и изключване на елемент не са допустими, се наричат **статични**, в противен случай - **динамични**.

В тази глава ще разгледаме структурата от данни масив и средствата, които я реализират.

Логическо описание

Масивът е крайна редица от фиксиран брой елементи от един и същ тип. Към всеки елемент от редицата е възможен пряк достъп, който се осъществява чрез индекс. Операциите включване и изключване на елемент в/от масива са недопустими, т.е. масивът е статична структура от данни.

Физическо представяне

Елементите на масива се записват последователно в паметта на компютъра, като за всеки елемент на редицата се отделя определено количество памет.

В езика C++ структурата масив се реализира чрез типа масив.

6.2 Тип масив

В C++ структурата от данни масив е реализирана малко ограничено. Разглежда се като крайна редица от елементи от един и същ тип с пряк достъп до всеки елемент, осъществяващ се чрез индекс с цели стойности, започващи от 0 и нарастващи с 1 до указана горна граница. Дефинира се от програмиста.

Дефиниране на масив

Типът масив се определя чрез задаване на типа и броя на елементите на редицата, определяща масив. Нека T е име или дефиниция на произволен тип, различен от псевдоним, void и функционален. За типа T и константния израз от интегрален или изброен тип с положителна стойност size, T[size] е тип масив от size елемента от тип T. Елементите се индексират от 0 до size-1. T се нарича **базов тип** за типа масив, а size – горна граница.

Примери:

int[5] дефинира масив от 5 елемента от тип int, индексирани от 0 до 4;

double[10] дефинира масив от 10 елемента от тип double, индексирани от 0 до 9;

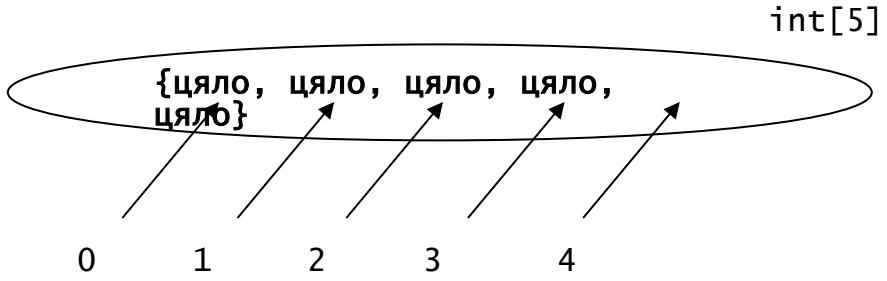
bool[4] дефинира масив от 4 елемента от тип bool, индексирани от 0 до 3.

Множество от стойности

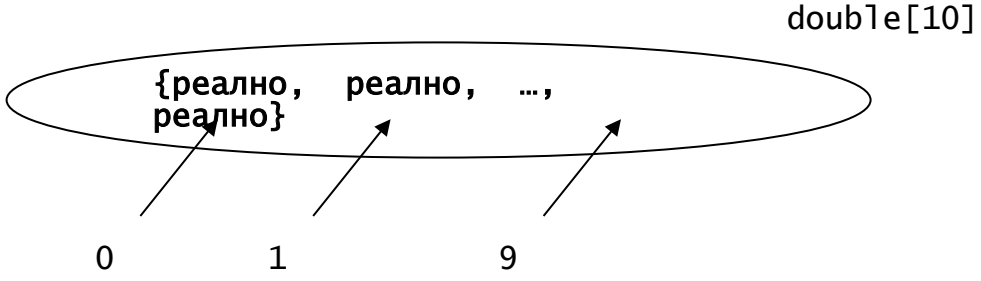
Множеството от стойности на типа T[size] се състои от всички редици от по size елемента, които са произволни константи от тип T. Достъпът до елементите на редиците е пряк и се осъществява с помощта на индекс, като достъпът до първия елемент се осъществява с индекс със стойност 0, до последния – с индекс със стойност size-1, а до всеки от останалите елементи – с индекс със стойност с 1 по-голяма от тази на индекса на предишния елемент.

Примери:

1. Множеството от стойности на типа int[5] се състои от всички редици от по 5 цели числа. Достъпът до елементите на редиците се осъществява с индекс със стойности 0, 1, 2, 3 и 4.



2. Множеството от стойности на типа double[10] се състои от всички редици от по 10 реални числа. Достъпът до елементите на редиците се осъществява с индекс със стойности 0, 1, 2, 3 и т.н. 9.



Елементите от множеството от стойности на даден тип масив са **константите** на този тип масив.

Примери:

1. Следните редици {1,2,3,4,5}, {-3, 0, 1, 2, 0}, {12, -14, 8, 23, 1000} са константи от тип int[5].

2. Редиците {1.5, -2.3, 3.4, 4.9, 5.0, -11.6, -123.56, 13.7, -32.12, 0.98}, {-13, 0.5, 11.9, 21.98, 0.03, 1e2, -134.9, 0.09, 12.3, 15.6} са константи от тип double[10].

Променлива величина, множеството от допустимите стойности на която съвпада с множеството от стойности на даден тип масив, се нарича променлива от дадения тип масив. Понякога ще я наричаме само масив.

Фиг. 6.1 определя дефиницията на променлива от тип масив. Тук общоприетият запис е нарушен. Променливата се записва между името на типа и размерността.

Дефиниция на масив

<дефиниция_на_променлива_от_тип_масив> ::=

T <променлива>[size] [= {<редица_от_константни_изрази>}]$_{опц}$

{,<променлива>[size] [= {<редица_от_константни_изрази>}]$_{опц}$ }$_{опц}$;

където

- T е име или дефиниция на произволен тип, различен от псевдоним, void, функционален;
- <променлива> е идентификатор;
- size е константен израз от интегрален или изброен тип с *положителна* стойност;
- <редица_от_константни_изрази> се дефинира по следния начин:

<редица_от_константни_изрази> ::= <константен_израз>|

<константен_израз>, <редица_от_константни_изрази>

като константните изрази са от тип T или от тип, съвместим с него.

Фиг. 6.1 Дефиниция на масив

Примери:

```
int a[5];
double c[10];
bool b[3];
enum {FALSE, TRUE} x[20];
double p[4] = {1.25, 2.5, 9.25, 4.12};
```

Дефиницията

<променлива>[size] = {<редица_от_константни_изрази>}

се нарича **дефиниция на масив с инициализация**, а фрагмента {<редица_от_константни_изрази>} - **инициализация**. При нея е възможно size да се пропусне. Тогава за стойност на size се подразбира броят на константните изрази, изброени в инициализацията. Ако size е указано и изброените константни изрази в инициализацията са по-малко от size, останалите се приемат за 0.

Примери:

1. Дефиницията

```
int q[5] =  {1, 2, 3};
```

е еквивалентна на

```
int q[] =  {1, 2, 3, 0, 0};
```

2. Дефиницията

```
double r[] = {0, 1, 2, 3};
```

е еквивалентна на

```
double r[4] = {0, 1, 2, 3};
```

Забележка: Не са възможни конструкции от вида:

```
int q[5];
q = {0, 1, 2, 3, 4};
```

а също

```
int q[];
```

и

```
double r[4] = {0.5, 1.2, 2.4, 1.2, 3.4};
```

Ще отбележим, че фрагментите

<променлива>[size] и

<променлива>[size] = {<редица_от_константни_изрази>}

от дефиницията от Фиг. 6.1 могат да се повтарят. За разделител се използва знакът запетая.

Пример: Дефиницията

```
double m1[20], m2[35], proben[30];
```

е еквивалентна на дефинициите:

```
double m1[20];
double m2[35];
double proben[30];
```

Дефиницията с инициализация е един начин за свързване на променлива от тип масив с конкретна константа от множеството от стойности на този тип масив. Друг начин предоставят т.нар. **индексирани променливи**. С всяка променлива от тип масив е свързан набор от индексирани променливи. Фиг. 6.2 илюстрира техния синтаксис.

Синтаксис на индексирани променлии

```
<индексирана_променлива> ::=
        <променлива_от_тип_масив>[<индекс>]
```

където

<индекс> е израз от интегрален или изброен тип.

Всяка индексирана променлива е от базовия тип.

Фиг. 6.2 Синтаксис на индексираните променливи

Примери:

1. С променливата a, дефинирана по-горе, са свързани индексираните променливи a[0], a[1], a[2], a[3] и a[4], които са от тип int.
2. С променливата b са свързани индексираните променливи b[0], b[1],…, b[9], които са от тип double.
3. С променливата x са свързани индексираните променливи x[0], x[1],…, x[19], които са от тип enum {FALSE, TRUE}.

Дефиницията на променлива от тип масив не само свързва променливата с множеството от стойности на указания тип, но и отделя определено количество памет (обикновено 4B), в която записва адреса в паметта на първата индексирана променлива на масива. Останалите индексирани променливи се разполагат последователно след първата. За всяка индексирана променлива се отделя по толкова памет, колкото базовият тип изисква.

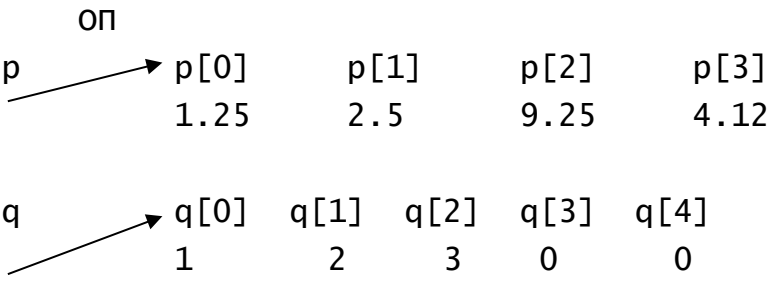
Пример:

ОП

a	a[0]	a[1]	…	a[4]	b	b[0]	b[1]	…	b[9]	…
адрес на a[0]	-	-		-	адрес на b[0]	-	-		-	
4в	4в	4в	…	4в	4в	8в	8в	…	8в	

За краткост, вместо “адрес на a[0]” ще записваме стрелка от a към a[0]. Стойността на отделената за индексираните променливи памет е неопределено освен ако не е зададена дефиниция с инициализация. Тогава в клетките се записват инициализиращите стойности.

Пример: Разпределението на паметта за променливите p и q, дефинирани в примерите по-горе, е следното:



Операции и вградени функции

Не са възможни операции над масиви като цяло, но всички операции и вградени функции, които базовият тип допуска, са възможни за индексираните променливи, свързани с масива.

Пример: Нека

```
int a[5], b[5];
```

Недопустими са:

```
cin >> a >> b;
a = b;
```

а също a == b или a != b.

Операторът

```
cout << a;
```

е допустим и извежда адреса на a[0].

Задачи върху тип масив

Задача 48. Да се напише програма, която въвежда последователно n числа, след което ги извежда в обратен ред.

Програма Zad48.cpp решава задачата.

```
// Program Zad48.cpp
#include <iostream.h>
int main()
{double x[100];
 cout << "n= ";
 int n;
 cin >> n; // въвеждане на стойност за n
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (n < 0 || n > 100)
 {cout << "Incorrect input! \n";
  return 1;
 }
 // n е цяло число от интервала [1, 100]
 // въвеждане на стойности за елементите на масива x
 for (int i = 0; i <= n-1; i++)
 {cout << "x[" << i << "]= ";
  cin >> x[i];
  if (!cin)
  {cout << "Error. Bad Input! \n";
   return 1;
  }
 } // извеждане на елементите на x в обратен ред
 for (i = n-1; i >= 0; i--)
    cout << x[i] << "\n";
 return 0;
}
```

Изпълнение на програма Zad48.cpp

Дефиницията double x[100]; води до отделяне на 800В ОП, които се именуват последователно с x[0], x[1], …, x[99] и са с неопределени стойности. Освен това се отделят 4В ОП за променливата x, в които се записва адресът на индексираната променлива x[0]. Следващият програмен фрагмент въвежда стойност на n (броя на елементите на масива, които ще бъдат използвани). Операторът

```
for (int i = 0; i <= n-1; i++)
 {cout << "x[" << i << "]= ";
  cin >> x[i];
  if (!cin)
  {cout << "Error. Bad Input! \n";
   return 1;
  }
 }
```

въвежда стойности на целите променливи x[0], x[1], …, x[n-1]. Всяка въведена стойност е предшествана от подсещане. Операторът

```
for (i = n-1; i >= 0; i--)
    cout << x[i] << "\n";
```

извежда в обратен ред компонентите на масива x.

Забележка: Фрагментите:

```
 …                          и        …
 cout << "n= ";                     int n = 10;
 int n;                             int x[10];
 cin >> n;                          …
 int x[n];
 …
```

са недопустими, тъй като n не е константен израз. Фрагментът

```
const int n = 10;
double x[n];
```

е допустим.

6.3 Някои приложения на структурата от данни масив

Търсене на елемент в редица

Нека са дадени редица от елементи a_0, a_1, …, a_{n-1}, елемент x и релация r. Могат да се формулират две основни задачи, свързани с търсене на елемент в редицата, който да е в релация r с елемента x.

а) Да се намерят **всички елементи** на редицата, които са в релация r с елемента x.

б) Да се установи, **съществува ли елемент** от редицата, който е в релация r с елемента x.

Съществуват редица методи, които решават едната, другата или и двете задачи. Ще разгледаме метода на **последователното търсене**, чрез който магат да се решат и двете задачи. Методът се състои в следното: последователно се обхождат елементите на редицата и за всеки елемент се проверява дали е в релация r с елемента x. При първата задача процесът продължава до изчерпване на редицата, а при втората – до намиране на първия елемент a_k (k = 0, 1, …, n-1), който е в релация r с x, или до изчерпване на редицата без да е намерен елемент с търсеното свойство.

Следващите четири задачи илюстрират този метод.

Задача 49. Дадени са редицата от цели числа a_0, a_1, …, a_{n-1} ($n \geq 1$) и цялото число x. Да се напише програма, която намира колко пъти x се съдържа в редицата.

В случая релацията r е операцията сравнение за равенство, която се реализира чрез оператора ==. Налага се всеки елемент на редицата да бъде сравнен с x, т.е. имаме задача от първия вид. Тя описва индуктивен цикличен процес.

Програма Zad49.cpp решава задачата.

```
// Program Zad49.cpp
#include <iostream.h>
int main()
{int a[20];
 cout << "n= ";
 int n;
 cin >> n; // въвеждане на дължината на редицата
```

```
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (n < 1 || n > 20)
 {cout << "Incorrect input! \n";
  return 1;
 }
 // въвеждане на редицата
 int i;
 for (i = 0; i <= n-1; i++)
 {cout << "a[" << i << "]= ";
  cin >> a[i];
  if (!cin)
  {cout << "Error. Bad input! \n";
   return 1;
  }
 }
 // въвеждане на стойност за x
 int x;
 cout << "x= ";
 cin >> x;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 // намиране на броя br на срещанията на x в редицата
 int br = 0;
 for (i = 0; i <= n-1; i++)
   if (a[i] == x) br++;
 cout << "number = " << br << "\n";
 return 0;
}
```

Задача 50. Дадени са редицата от цели числа a_0, a_1, …, a_{n-1} ($n \geq 1$) и цялото число x. Да се напише програма, която проверява дали x се съдържа в редицата.

В този случай се изисква при първото срещане на елемент от редицата, който е равен на x, да се преустанови работата с подходящо съобщение. Броят на сравненията на x с елементите от редицата е ограничен отгоре от n, но не е известен.

Програма Zad50.cpp решава задачата. Фрагментът, реализиращ входа, е същия като в Zad49.cpp и затова е пропуснат.

```
// Program Zad50.cpp
#include <iostream.h>
int main()
{int a[20];
 …
 i = 0;
 while (a[i] != x && i < n-1)
    i++;
 if (a[i] == x) cout << "yes \n";
 else cout << "no \n";
 return 0;
}
```

Обхождането на редицата става чрез промяна на стойностите на индекса i - започват от 0 и на всяка стъпка от изпълнението на тялото на цикъла се увеличават с 1. Максималната им стойност е n-1. При излизането от цикъла ще е в сила отрицанието на условието (a[i] != x && i < n-1), т.е. (a[i] == x || i == n-1). Ако е в сила a[i] == x, тъй като сме осигурили a[i] да е елемент на редицата, отговорът “yes” е коректен. В противен случай е в сила i == n-1, т.е. сканиран е и последният елемент на редицата и за него не е вярно a[i] == x. Това е реализирано чрез отговора “no” от алтернативата на условния оператор.

Фрагментът

```
i = -1;
do
```

```
   i++;
   while (a[i] != x && i < n-1);
   if (a[i] == x) cout << "yes \n";
   else cout << "no \n";
```
реализира търсенето чрез използване на оператора do/while.

Задача 51. Да се напише програма, която установява, дали редицата от цели числа a_0, a_1, ..., a_{n-1} е монотонно намаляваща.

а) За решаването на задачата е необходимо да се установи, дали за всяко i (0 ≤ i ≤ n-2) е в сила релацията a[i] >= a[i+1]. Това може да се реализира като се провери дали броят на целите числа i (0≤i≤n-2), за които е в сила релацията a[i] ≥ a[i+1], е равен на n-1.

Програмата Zad51_1.cpp реализира този начин за проверка дали редица е монотонно намаляваща. Фрагментите, реализиращи въвеждането на n и масива a, са известни вече и затова са пропуснати.

```
// Program Zad51_1.cpp
#include <iostream.h>
int main()
{int a[100];
 // дефиниране и въвеждане на стойност на n
 …
 // въвеждане на масива a
 …
 int br = 0;
 for (i = 0; i <= n-2; i++)
    if (a[i] >= a[i+1]) br++;
 if (br == n-1) cout << "yes \n";
 else cout << "no \n";
 return 0;
}
```

б) Задачата може да се сведе до търсене на i (i = 0, 1,..., n-2), така че a[I] < a[i+1], т.е. до задача за съществуване.

Програма Zad51_2.cpp реализира този начин за проверка дали редица е монотонно намаляваща. Фрагментите, реализиращи въвеждането на n и масива a отново са пропуснати.

```
// Program Zad51_2.cpp;
#include <iostream.h>
int main()
{int a[100];
 //въвеждане на размерността n и масива a
 …
 i = 0;
 while (a[i] >= a[i+1] && i < n-2) i++;
 if (a[i] >= a[i+1]) cout << "yes \n";
 else cout << "no \n";
 return 0;
}
```

Решение б) е по-ефективно, тъй като при първото срещане на a[i], така че релацията a[i] < a[i+1] е в сила, изпълнението на цикъла while завършва. Решение а) реализира последователно търсене е пълно изчерпване, а решение б) – задача за съществуване на елемент в редица, който е в определена релация с друг елемент (в случая съседния му).

Задача 52. Да се напише програма, която установява, дали редицата от цели числа a_0, a_1, ..., a_{n-1} се състои от различни елементи.

а) За решаването на задачата е необходимо да се установи, дали за всяка двойка (i, j): $0 \le i \le n-2$ и $i+1 \le j \le n-1$ е в сила релацията a[i] != a[j]. Това може да се постигне като се провери дали броят на двойките (i, j): $0 \le i \le n-2$ и $i+1 \le j \le n-1$, за които е в сила релацията a[i] != a[j], е равен на n*(n-1)/2.

Програма Zad52_1.cpp реализира тази идея за решение на задачата – търсене с пълно изчерпване. Фрагментите, реализиращи въвеждането на n и масива a, отново са пропуснати.

```
// Program Zad52_1.cpp
```

```
#include <iostream.h>
int main()
{int a[100];
 //въвеждане на размерността n и масива a
 …
 int br = 0;
 for (i = 0; i <= n-2; i++)
    for (int j = i+1; j <= n-1; j++)
       if (a[i] != a[j]) br++;
 if (br == n*(n-1)/2) cout << "yes \n";
 else cout << "no \n";
 return 0;
}
```

б) Задачата може да се сведе до проверка за съществуване на двойка индекси (i, j): 0 ≤ i ≤ n-2 и i+1 ≤ j ≤ n-1, за които не е в сила релацията a[i] != a[j]. Програма Zad52_2.cpp реализира тази идея.

```
// Program Zad52_2.cpp
#include <iostream.h>
int main()
{int a[100];
 // въвеждане стойности на размерността n и масива a
 …
 i = -1;
 int j;
 do
 {i++;
  j = i+1;
  while (a[i] != a[j] && j < n-1) j++;
 } while (a[i] != a[j] && i < n-2);
 if (a[i] != a[j]) cout << "yes \n";
 else cout << "no \n";
 return 0;
}
```

Решение б) е по-ефективно, тъй като при първото срещане на a[i] и a[j], така че релацията a[i] == a[j] е в сила, изпълнението на

операторите за цикъл завършва. То реализира задача за съществуване на метода за търсене.

Сортиране на редица

Да се сортира редицата a_0, a_1, ..., a_{n-1} означава така да се пренаредят елементите ѝ, че за новата редица да е в сила $a_0 \le a_1 \le ... \le a_{n-1}$ или $a_0 \ge a_1 \ge ... \ge a_{n-1}$. В първия случай се казва, че редицата е сортирана във **възходящ**, а във втория случай, че е сортирана в **низходящ ред**.

Съществуват много методи за сортиране на редици от елементи. В тази глава ще разгледаме **метода на пряката селекция** и чрез него ще реализираме възходяща сортировка на редица.

Метод на пряката селекция

Разглежда се редицата a_0, a_1, …, a_{n-1} и се извършват следните действия:

- Намира се k, така че $a_k = \min\{a_0, a_1, …, a_{n-1}\}$.
- Разменят се стойностите на a_k и a_0.

Така на първо място в редицата се установява най-малкият ѝ елемент.

Разглежда се редицата a_1, a_2, …, a_{n-1} и се извършват действията:

- Намира се k, така че $a_k = \min\{a_1, a_2, …, a_{n-1}\}$.
- Разменят се стойностите на a_k и a_1.

Така на второ място в редицата се установява следващият по големина елемент на редицата и т.н.

Разглежда се редицата a_{n-2}, a_{n-1} и се извършват действията:

- Намира се k, така че $a_k = \min\{a_{n-2}, a_{n-1}\}$.
- Разменят се стойностите на a_k и a_{n-2}.

Получената редица е сортирана във възходящ ред.

Задача 53. Да се сортира във възходящ ред по метода на пряката селекция числовата редица a_0, a_1, …, a_{n-1} ($n \ge 1$).

Програма Zad53.cpp решава задачата. Фрагментът за въвеждане стойности на размерността n и масива a отново е пропуснат.

```
// Program Zad53.cpp
#include <iostream.h>
#include <iomanip.h>
int main()
{int a[100];
 ...
 int i;
 for (i = 0; i <= n-2; i++)
 {int min = a[i];
  int k = i;
  for (int j = i+1; j <= n-1; j++)
    if (a[j] < min)
    {min = a[j];
     k = j;
    }
  int x = a[i]; a[i] = a[k]; a[k] = x;
 }
 for (i = 0; i <= n-1; i++)
   cout << setw(10) << a[i];
 cout << '\n';
 return 0;
}
```

Сливане на редици

Сливането е вид сортиране. Нека са дадени сортираните във възходящ ред редици:

$a_0, a_1, \ldots, a_{n-1}$

$b_0, b_1, \ldots, b_{m-1}$

Да се слеят редиците означава да се конструира нова, сортирана във възходящ ред редица, съставена от елементите на дадените редици. Осъществява се по следния начин:

- Поставят се “указатели” към първите елементи на редиците $\{a_i\}$ и $\{b_j\}$.

- Докато има елементи и в двете редици, се сравняват елементите, сочени от “указателите”. По-малкият елемент се записва в новата редица, след което се прескача.

- След изчерпване на елементите на едната от дадените редици, елементите на другата от “указателя” (включително) се прехвърлят в новата редица.

Задача 54. Дадени са сортираните във възходящ ред редици:

$a_0, a_1, \ldots, a_{n-1}$

$b_0, b_1, \ldots, b_{m-1}$

($n \geq 1$, $m \geq 1$). Да се напише програма, която слива двете редици в редицата

$c_0, c_1, \ldots, c_{k-1}$.

Програма Zad54.cpp решава задачата.

```
// Program Zad54.cpp
#include <iostream.h>
#include <iomanip.h>
int main()
{int a[20];
 cout << "n= ";
 int n;
 cin >> n;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (n < 1 || n > 20)
 {cout << "Incorrect input! \n";
  return 1;
 }
 int i;
 for (i = 0; i <= n-1; i++)
 {cout << "a[" << i << "]= ";
  cin >> a[i];
 }
```

```
int b[10];
cout << "m= ";
int m;
cin >> m;
if (!cin)
{cout << "Error. Bad input! \n";
 return 1;
}
if (m < 1 || m > 10)
{cout << "Incorrect input! \n";
 return 1;
}
for (i = 0; i <= m-1; i++)
{cout << "b[" << i << "]= ";
 cin >> b[i];
}
int p1 = 0, p2 = 0;
int c[30];
int p3 = -1;
while (p1 <= n-1 && p2 <= m-1)
if (a[p1] <= b[p2])
{p3++;
 c[p3] = a[p1];
 p1++;
}
else
{p3++;
c[p3] = b[p2];
p2++;
}
if (p1 > n-1)
   for (i = p2; i <= m-1; i++)
    {p3++;
     c[p3] = b[i];
    }
else
```

```
    for (i = p1; i <= n-1; i++)
     {p3++;
      c[p3] = a[i];
     } // извеждане на редицата
  for (I = 0; i <= p3; i++)
    cout << setw(10) << c[i];
  cout << '\n';
  return 0;
 }
```

Разглежданите досега масиви се наричат **едномерни**. Те реализират крайни редици от елементи от скаларен тип. Възможно е обаче типът на елементите да е масив. В този случай се говори за многомерни масиви.

6.4 Многомерни масиви

Масив, базовият тип на който е едномерен масив, се нарича **двумерен**. Масив, базовият тип на който е двумерен масив, се нарича **тримерен** и т.н. На практика се използват масиви с размерност най-много 3.

Дефиниране на многомерни масиви

Нека T е име или дефиниция на произволен тип, различен от псевдоним, void и функционален, $size_1$, $size_2$, …, $size_n$ (n>1 е дадено цяло число) са константни изрази от интегрален или изброен тип с положителни стойности. T[$size_1$][$size_2$] … [$size_n$] е тип n-мерен масив от тип T. T се нарича базов тип за типа масив.

Примери:

int [5][3] дефинира двумерен масив от тип int;

double [4][5][3] дефинира тримерен масив от тип double;

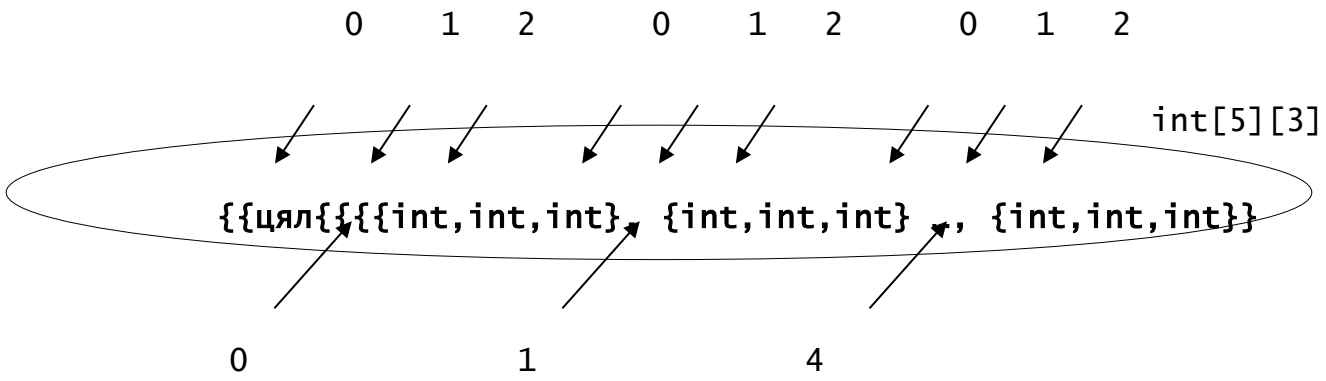
Множество от стойности

Множеството от стойности на типа T[$size_1$][$size_2$] … [$size_n$] се състои от всички редици от по $size_1$ елемента, които са произволни

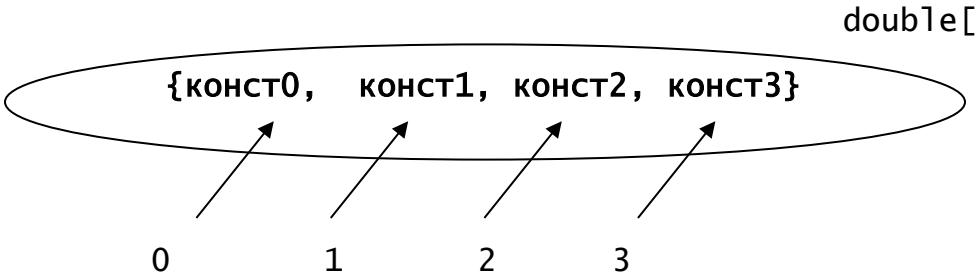
константи от тип T[size$_2$] … [size$_n$]. Достъпът до елементите на редиците е пряк и се осъществява с помощта на индекс, като достъпът до първия елемент се осъществява с индекс със стойност 0, до последния – с индекс със стойност size$_1$-1, а до всеки от останалите елементи – с индекс със стойност с 1 по-голяма от тази на индекса на предишния елемент. Елементите от множеството от стойности на даден тип многомерен масив са **константите** на този тип масив.

Примери:

1. Множеството от стойности на типа int[5][3] се състои от всички редици от по 5 елемента, които са едномерни масиви от тип int[3]. Достъпът до елементите на редиците се осъществява с индекс със стойности 0, 1, 2, 3 и 4.



2. Множеството от стойности на типа double[4][5][3] се състои от всички редици от по 4 константи от тип double[5][3]. Достъпът до елементите на редиците се осъществява с индекс със стойности 0, 1, 2 и 3.



където с конст$_i$ (i = 0, 1, 2, 3) е означена произволна константа от тип double[5][3].

Променлива величина, множеството от допустимите стойности на която съвпада с множеството от стойности на даден тип масив, се нарича променлива от дадения тип масив или само масив. Фиг. 6.3 дава обобщение на синтаксиса на дефиницията на променлива от тип масив.

Синтаксис на многомерен масив

<дефиниция_на_променлива_от_тип_многомерен_масив> ::=

T <променлива>[$size_1$][$size_2$]…[$size_n$]

[= {<редица_от_константи_от_тип T1>}]$_{опц}$

{,<променлива>[$size_1$][$size_2$]…[$size_n$]

[= {<редица_от_константи_от_тип T1>}]$_{опц}$}$_{опц}$;

{,<променлива>[$size_1$][$size_2$]…[$size_n$]

[= {<редица_от_константи_от_тип T>}]$_{опц}$}$_{опц}$;

където

- T е име или дефиниция на произволен тип, различен от псевдоним, void и функционален;
- T1 е име на типа T[$size_2$]…[$size_n$];
- $size_1$, $size_2$,…, $size_n$ са константни изрази от интегрален или изброен тип със *положителни* стойности;
- <променлива> е идентификатор;
- <редица_от_константи_от_тип T1> се дефинира по следния начин:

<редица_от_константи_от_тип T1> ::= <константа_от_тип T1>|

<константа_от_тип T1>, <редица_от_константи_от_тип T1>

- <редица_от_константи_от_тип T> се определя по аналогичен начин.

Фиг. 6.3 Синтаксис на многомерен масив

Примери:

```
int x[10][20];
double y[20][10][5];
int z[3][2] = {{1, 3},
               {5, 7},
                  {2, 9}};
int t[2][3][2] = {{{1, 3}, {5, 7}, {6, 9}},
                  {{7, 8}, {1, 8}, {-1, -4}}};
```

Ще отбележим, че фрагментите:
<променлива>[size$_1$][size$_2$] … [size$_n$],
<променлива>[size$_1$][size$_2$]…[size$_n$]={<редица_от_константи_от_тип T1>}
<променлива>[size$_1$][size$_2$]…[size$_n$]={<редица_от_константи_от_тип T>}
от дефиницията от Фиг. 6.3, могат да се повтарят. За разделител се използва символът запетая.

Примери:

```
int a[3][4], b[2][3][2] = {{{1, 2}, {3, 4}, {5, 6}},
                           {{7, 8}, {9, 0}, {1, 2}}};
double c[2][3] = {1, 2, 3, 4, 5, 6}, d[3][4][5][6];
```

При дефиницията с инициализация, от Фиг. 6.3, е възможно size$_1$ да се пропусне. Тогава за стойност на size$_1$ се подразбира броят на редиците от константи на най-външно ниво, изброени при инициализацията.

Пример: Дефиницията

```
int s[][2][3] = {{{1,2,3}, {4, 5, 6}},
                 {{7, 8, 9}, {10, 11, 12}},
                 {{13, 14, 15}, {16, 17, 18}}};
```

е еквивалентна на

```
int s[3][2][3] = {{{1,2,3}, {4, 5, 6}},
                  {{7, 8, 9}, {10, 11, 12}},
                  {{13, 14, 15}, {16, 17, 18}}};
```

Ако изброените константни изрази в инициализацията на ниво i са по-малко от size$_i$, останалите се инициализират с нулеви стойности.

Примери: Дефиницията

```
int fi[5][6] = {{1, 2}, {5}, {3, 4, 5},
                {2, 3, 4, 5}, {2, 0, 4}};
```

е еквивалентна на

```
int fi[5][6] = {{1, 2, 0, 0, 0, 0},
                {5, 0, 0, 0, 0, 0},
                {3, 4, 5, 0, 0, 0},
                {2, 3, 4, 5, 0, 0},
                {2, 0, 4, 0, 0, 0}};
```

Вложените фигурни скоби не са задължителни. Следното инициализиране

```
int ma[4][3] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
```

е еквивалентно на

```
   int ma[4][3] = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}, {9, 10, 11}};
```
но е по-неясно.

Следващата дефиниция
```
   int ma[4][3] = {{0}, {1}, {2}, {3}};
```
е еквивалентна на
```
   int ma[4][3] = {{0, 0, 0 }, {1, 0, 0}, {2, 0, 0}, {3, 0, 0}};
```
и е различна от
```
   int ma[4][3] = {0, 1, 2, 3};
```
която пък е еквивалентна на
```
   int ma[4][3] = {{0, 1, 2}, {3, 0, 0}, {0, 0, 0}, {0, 0, 0}};
```

Инициализацията е един начин за свързване на променлива от тип масив с конкретна константа от множеството от стойности на този тип масив. Друг начин дават индексираните променливи. С всяка променлива от тип масив е свързан набор от индексирани променливи. Фиг. 6.4 обобщава техния синтаксис.

Синтаксис на индексирана променлива от тип многомерен масив

<индексирана_променлива> ::=

<променлива_от_тип_масив>[<индекс$_1$>][<индекс$_2$>] … [<индекс$_n$>]

където

<индекс$_i$> е *израз* от интегрален или изброен тип.

Всяка индексирана променлива е от базовия тип.

Фиг. 6.4 Синтаксис на индексирана променлива от тип многомерен масив

Примери:

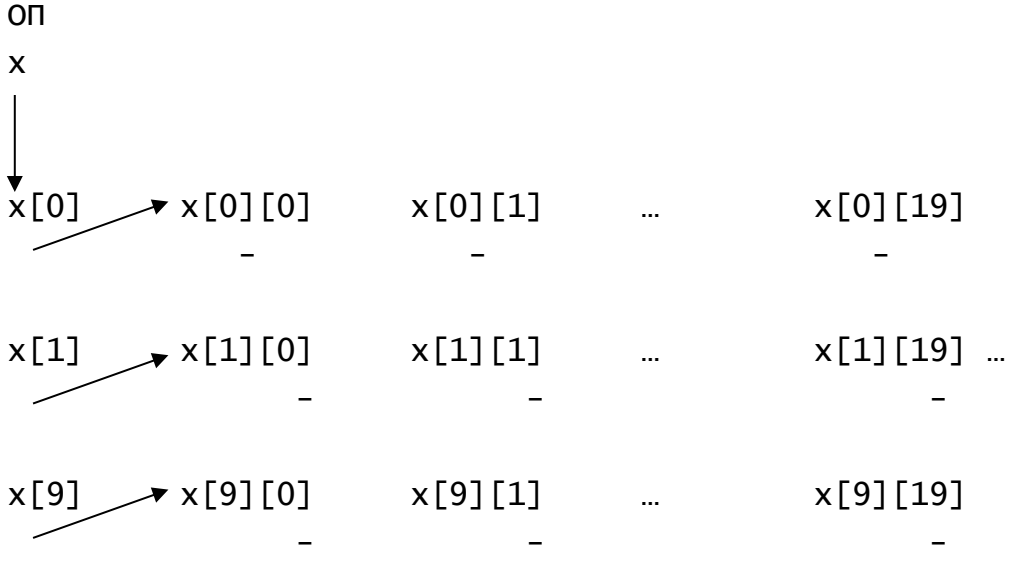
1. С променливата x, дефинирана по-горе, са свързани индексираните променливи
```
   x[0][0],    x[0][1],    …, x[0][19],
   x[1][0],    x[1][1],    …, x[1][19],
                           …
   x[9][0],    x[9][1],    …, x[9][19],
```
които са от тип int.

2. С променливата y са свързани следните реални индексирани променливи:

y[i][0][0], y[i][0][1], …, y[i][0][4],
y[i][1][0], y[i][1][1], …, y[i][1][4],
…
y[i][9][0], y[i][9][1], …, y[i][9][4],
за i = 0, 1, …, 19.

Дефиницията на променлива от тип многомерен масив не само свързва променливата с множеството от стойности на указания тип, но и отделя определено количество памет за разполагане на индексираните променливи, свързани с нея. Индексираните променливи се разполагат последователно по по-бързото нарастване на по-далечните си индекси. За всяка индексирана променлива се отделя толкова памет, колкото базовият тип изисква. Чрез пример ще илюстрираме по-подробно представянето.

След дефиницията
int x[10][20];
за променливата x се отделят 4B, в които се записва адресът на първата компонента, в случая x[0], на едномерен масив, елементите на който са адреси. Стойността на индексираната променлива x[i] е адреса на индексираната променлива x[i][0] (i = 0, 1, ..., 9), т.е.



Операторът
cout << x[0] << x[1] << ... << x[9];
ще изведе адресите на x[0][0], x[1][0], ... и x[9][0].
Индексираните променливи x[i][j] (i=0, 1, ..., 9; j = 0, 1, ..., 19) са променливи от тип int и за всяка от тях са отделени 4B ОП, които

са с неопределени стойности, тъй като x не е дефинирана с инициализация.

Забележка: Двумерните масиви разполагат в ОП индексираните си променливи по по-бързото нарастване на втория индекс. Това физическо представяне се нарича **представяне по редове**. Тези масиви могат да бъдат използвани за реализация и работа с матрици и др. правоъгълни таблици.

Важно допълнение: При работа с масиви трябва да се има предвид, че повечето реализации не проверяват дали стойностите на индексите са в рамките на границите, зададени при техните дефиниции. Тази особеност крие опасност от допускане на трудноотриваеми грешки.

Често допускана грешка: В Паскал, Ада и др. процедурни езици, индексите на индексираните променливи се ограждат само в една двойка квадратни скоби и се отделят със запетаи. По навик, при програмиране на C++, често се използва същото означение. Това е неправилно, но за съжаление не винаги е съпроводено със съобщение за грешка, тъй като в езика C++ съществуват т.нар. comma-изрази. Използвахме ги вече в заглавните части на оператора за цикъл for. Comma-изразите са изрази, отделени със запетаи. Стойността на най-десния израз е стойността на comma-израза. Операторът за последователно изпълнение запетая е лявоасоциативен. Така 1+3, 8, 21-15 е comma-израз със стойност 6, а [1, 2] е comma-израз със стойност [2]. В C++ ma[1,2] означава адреса на индексираната променлива ma[2][0] (индексът [0] се добавя автоматично).

Задачи върху многомерни масиви

Задача 55. Да се напише програма, която въвежда елементите на правоъгълна матрица a[nxm] от цели числа и намира и извежда матрицата, получена от дадената като всеки от нейните елементи е увеличен с 1.

Програма Zad55.cpp решава задачата.

```
// Program Zad55.cpp
#include <iostream.h>
#include <iomanip.h>
```

```
int main()
{int a[10][20];
 // въвеждане на броя на редовете на матрицата
 cout << "n= ";
 int n;
 cin >> n;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (n < 1 || n > 10)
 {cout << "Incorrect input! \n";
  return 1;
 }
// въвеждане на броя на стълбовете на матрицата
 cout << "m= ";
 int m;
 cin >> m;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (m < 1 || m > 20)
 {cout << "Incorrect input! \n";
  return 1;
 }
 // въвеждане на матрицата по редове
 int i, j;
 for (i = 0; i <= n-1; i++)
    for (j = 0; j <= m-1; j++)
    {cout << "a[" << i << ", " << j << "]= ";
     cin >> a[i][j];
     if (!cin)
     {cout << "Error. Bad Input! \n";
      return 1;
     }
```

```
      }
   // конструиране на нова матрица b
   int b[10][20];
   for (i = 0; i <= n-1; i++)
      for (j = 0; j <= m-1; j++)
         b[i][j] = a[i][j] + 1;
   // извеждане на матрицата b по редове
   for (i = 0; i <= n-1; i++)
   {for (j = 0; j <= m-1; j++)
     cout << setw(6) << b[i][j];
     cout << '\n';
   }
   return 0;
  }
```

Забележка: За реализиране на операциите въвеждане, извеждане и конструиране се извърши обхождане на елементите на двумерен масив по редове.

Задача 56. Да се напише програма, която намира и извежда сумата от елементите на всеки стълб на квадратната матрица a[nxn].

Програма Zad56.cpp решава задачата.

```
// Program Zad56.cpp
#include <iostream.h>
#include <iomanip.h>
int main()
{int a[10][10];
 cout << "n= ";
 int n;
 cin >> n;
 if (!cin)
 {cout << "Error. Bad input! \n";
  return 1;
 }
 if (n < 1 || n > 10)
 {cout << "Incorrect input! \n";
```

```
     return 1;
    }
    int i, j;
    for (i = 0; i <= n-1; i++)
       for (j = 0; j <= n-1; j++)
       {cout << "a[" << i << ", " << j << "]= ";
        cin >> a[i][j];
        if (!cin)
        {cout << "Error. Bad Input! \n";
         return 1;
        }
       }
    for (j = 0; j <= n-1; j++)
    {int s = 0;
     for (i = 0; i <= n-1; i++)
       s += a[i][j];
     cout << setw(10) << j << setw(10) << s << "\n";
    }
    return 0;
   }
```

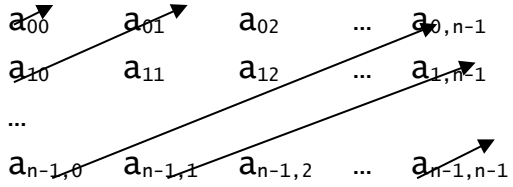
Реализирано е обхождане на масива по стълбове (първият индекс се изменя по-бързо).

Задача 57. Да се напише програмен фрагмент, който намира номерата на редовете на целочислената квадратна матрица a[nxn], в които има елемент, равен на цялото число x.

```
 …
 for (i = 0; i <= n-1; i++)
 {j = -1;
  do
  j++;
  while (a[i][j] != x && j < n-1);
  if (a[i][j] == x) cout << setw(5) << i << '\n';
  }
 …
```

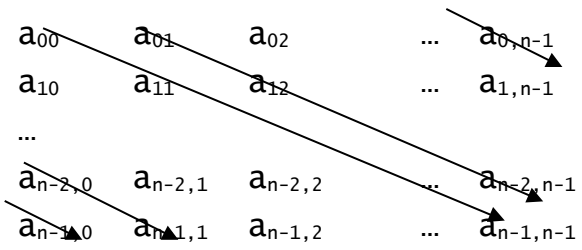
фрагментът реализира последователно обхождане на *всички* редове на матрицата и за всеки ред проверява дали *съществува* елемент, равен на дадения елемент x.

Задача 58. Да се напише програмен фрагмент, който обхожда квадратната матрица a[nxn] по диагонали, започвайки от елемента a_{00}, както е показано по-долу:




```
…
int k;
for (k = 0; k <= n-1; k++)
{for (i = k; i >= 0; i--)
 cout << "(" << i << ", "<< k-i << ") ";
 cout << '\n';
}
for (k = n; k <= 2*n-2; k++)
 {for (i = n-1; i >= k-n+1; i--)
  cout << "(" << i << ", "<< k-i << ") ";
  cout << '\n';
 }
…
```

Задача 59. Да се напише програмен фрагмент, който обхожда квадратната матрица a[nxn] по диагонали, започвайки от елемента $a_{n-1,0}$, както е показано по-долу:



```
 …
int k;
for (k = n-1; k >= 0; k--)
{for (i = k; i <= n-1; i++)
 cout << "(" << i << ", "<< i-k << ") ";
 cout << '\n';
}
for (k = -1; k >= 1-n; k--)
{for (i = 0; i <= n+k-1; i++)
 cout << "(" << i << ", "<< i-k << ") ";
 cout << '\n';
}
…
```

Задача 60. Да се напише програма, която:

а) въвежда по редове елементите на квадратната реална матрица A с размерност nxn;

б) от матрицата A конструира редицата B: b_0, b_2,..., b_{m-1}, където m = n^2, при което първите n елемента на B съвпадат с елементите на първия стълб на A, вторите n елемента на B съвпадат с елементите на втория стълб на A и т.н., последните n елемента на B съвпадат с елементите на последния стълб на A;

в) сортира във възходящ ред елементите на редицата B;

г) образува нова квадратна матрица A с размерност nxn, като елементите от първия ред на A съвпадат с първите n елемента на B, елементите от втория ред на A съвпадат с вторите n елемента на B и т.н. елементите от n - тия ред на A съвпадат с последните n елемента на B;

д) извежда по редове новата матрица A.

Програма Zad60.cpp решава задачата.

```
// Program Zad60.cpp
#include <iostream.h>
#include <iomanip.h>
int main()
{int a[10][10];
```

```
cout << "n= ";
int n;
cin >> n;
if (!cin)
{cout << "Error. Bad input! \n";
 return 1;
}
if (n < 1 || n > 10)
{cout << "Incorrect input! \n";
 return 1;
}
// въвеждане на масива a
int i, j;
for (i = 0; i <= n-1; i++)
    for (j = 0; j <= n-1; j++)
   {cout << "a[" << i<< "][" << j  << "]= ";
    cin >> a[i][j];
   }
// извеждане на елементите на a по редове
for (i = 0; i <= n-1; i++)
{for (j = 0; j <= n-1; j++)
  cout << setw(5) << a[i][j];
 cout << "\n";
}
// развиване на матрицата a по стълбове
int b[100];
int m = -1;
for (j = 0; j <= n-1; j++)
   for (i = 0; i <= n-1; i++)
   {m++;
    b[m] = a[i][j];
   }
   m++; // m е броя на елементите на редицата b
// извеждане на редицата b
for (i = 0; i <= m-1; i++)
   cout << setw(5) << b[i];
```

```
 cout << '\n';
 // сортиране на b по метода на пряката селекция
 for (i = 0; i <= m-2; i++)
 {int k = i;
  int min = b[i];
  for (j = i+1; j <= m-1; j++)
    if (b[j] < min)
    {min = b[j];
     k = j;
    }
  int x = b[i]; b[i] = b[k]; b[k] = x;
 }
 // извеждане на сортираната b
 for (i = 0; i <= m-1; i++)
    cout << setw(5) << b[i];
 cout << '\n';
 // конструиране на новата матрица a
 m = -1;
 for (i = 0; i <= n-1; i++)
    for (j = 0; j <= n-1; j++)
    {m++;
    a[i][j] = b[m];
    }
 // извеждане на матрицата a
 for (i = 0; i <= n-1; i++)
   {for (j = 0; j <= n-1; j++)
      cout << setw(10) << a[i][j];
    cout << '\n';
   }
 return 0;
}
```

6.5 Символни низове

6.5.1. Структура от данни низ

Логическо описание

Крайна, евентуално празна редица от символи, заградени в кавички, се нарича **символен низ**, **знаков низ** или само **низ**.

Броят на символите в редицата се нарича **дължина** на низа. Низ с дължина 0 се нарича **празен**.

Примери: “xyz” е символен низ с дължина 3,
“This is a string.” е символен низ с дължина 17, а
“” е празния низ.

Низ, който се съдържа в даден низ се нарича негов **подниз**.

Пример: Низът “ is s “ е подниз на низа “This is a string.”, а низът “ is a sing” не е негов подниз.

Конкатенация на два низа е низ, получен като в края на първия низ се запише вторият. Нарича се още **слепване** на низове.

Пример: Конкатенацията на низовете “a+b” и “=b+a” е низът “a+b=b+a”, а конкатенацията на “=b+a” с “a+b” е низът “=b+aa+b”. Забелязваме, че редът на аргументите е от значение.

Два символни низа се сравняват по следния начин: Сравнява се всеки символ от първия низ със символа от съответната позиция на втория низ. Сравнението продължава до намиране на два различни символа или до края на поне един от символните низове. Ако кодът на символ от първия низ е по-малък от кода на съответния символ от втория низ, или първият низ е изчерпен, приема се, че първият низ е по-малък от втория. Ако пък е по-голям или вторият низ е изчерпен – приема се, че първият низ е по-голям от втория. Ако в процеса на сравнение и двата низа едновременно са изчерпени, те са равни. Това сравнение се нарича **лексикографско**.

Примери: “abbc” е равен на “abbc”
“abbc” е по-малък от “abbcaaa”
“abbc” е по-голям от “aa”
“abbcc” е по-голям от “abbc”.

Физическо представяне

Низовете се представят последователно.

6.5.2 Символни низове в езика C++

Съществуват два начина за разглеждане на низовете в езика C++:

- като едномерни масиви от символи;
- като указатели към тип char.

За щастие, те са семантично еквивалентни.

В тази част ще разгледаме символните низове като масиви от символи.

Дефиницията

```
char str1[100];
```

определя променливата str1 като масив от 100 символа, а

```
char str2[5] = {‘a’, ‘b’, ‘c’};
```

дефинира масива от символи str2 и го инициализира. Тъй като при инициализацията са указани по-малко от 5 символа, останалите се допълват с нулевия символ, който се означава със символа \0, а понякога и само с 0. Така последната дефиниция е еквивалентна на дефинициите:

```
char str2[5] = {‘a’, ‘b’, ‘c’, ‘\0’, ‘\0’};
char str2[5] = {‘a’, ‘b’, ‘c’, 0, 0};
```

Всички действия за работа с едномерни масиви, които описахме в т. 2 и 4, са валидни и за масиви от символи с изключение на извеждането. Операторът

```
cout << str2;
```

няма да изведе адреса на str2 (както е при масивите от друг тип), а текста

```
abc
```

Има обаче една особеност. Ако инициализацията на променливата str2 е пълна (съдържа точно 5 символа) и не завършва със символа \0, т.е. има вида

```
char str2[5] = {‘a’, ‘b’, ‘c’, ‘d’, ‘e’};
```

операторът

```
cout << str2;
```

извежда текста

```
abcde<неопределено>
```

Имайки пред вид това, бихме могли да напишем подходящи програмни фрагменти, които въвеждат, извеждат, копират, сравняват, извличат части, конкатенират низове. Тъй като операциите се извършват над

индексираните променливи, налага се да се поддържа целочислена променлива, съдържаща дължината на низа.

В езика са дадени средства, реализиращи низа като скаларна структура. За целта низът се разглежда като редица от символи, завършваща с нулевия символ \0, наречен още **знак за край на низ**. Тази организация има предимството, че не е необходимо с всеки низ да се пази в променлива дължината му, тъй като знакът за край на низ позволява да се определи краят му.

Примери: Дефинициите

```
char s1[5] = {‘a, ‘b’, ‘b’, ‘a’, ‘\0’};
char s2[10] = {‘x’, ‘y’, ‘z’, ‘1’, ‘2’, ‘+’, ‘\0’};
```

свързват променливите s1 и s2 от тип масив от символи с низовете “abba” и “xyz12+” съответно. Знакът за край на низ ‘\0’ не се включва явно в низа.

Този начин за инициализация не е много удобен. Следните дефиниции са еквивалентни на горните.

```
char s1[5] = “abba”;
char s2[10] = “xyz12+”;
```

Забелязваме, че ако низ, съдържащ n символа, трябва да се свърже с масив от символи, минималната дължина на масива трябва да бъде n+1, за да се поберат n-те символа и символът \0.

6.5.3 Тип символен низ

Дефиниране

Типът char[size], където size е константен израз от интегрален или изброен тип, **може да бъде използван за задаване на тип низ с максимална дължина size-1**.

Пример:

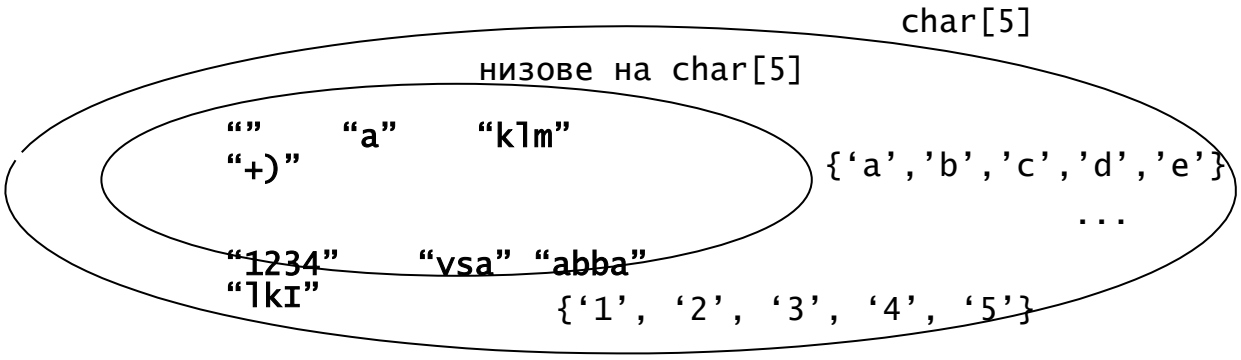
char[5] може да се използва за задаване на тип низ с максимална дължина 4.

Множество от стойности

Множеството от стойности на типа низ, зададен чрез char[size], се състои от всички низове с дължина 0, 1, 2, ..., size-1. То е подмножество на множеството от стойности на типа char[size].

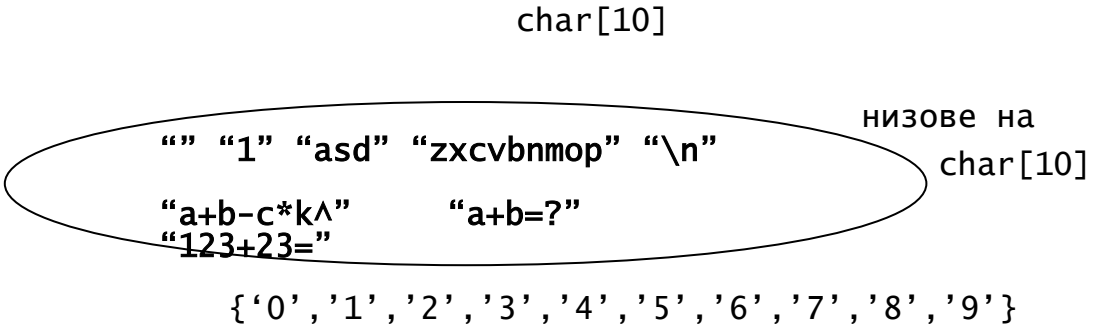
Примери:

1. Множеството от стойности на типа низ, зададен чрез char[5] се състои от всички низове с дължина 0, 1, 2, 3 и 4.



Забележка: Редицата {‘\0’, ‘\0’, ‘\0’, ‘\0’, ‘\0’} представя празния низ “”,{c, ‘\0’, ‘\0’, ‘\0’, ‘\0’}, където с е произволен символ представя низ с дължина 1, {c, c, ‘\0’, ‘\0’, ‘\0’} е низ с дължина 2 и т.н.

2. Множеството от стойности на типа char[10] се състои от всички низове с дължина 0, 1, 2, ..., 9.



Елементите от множеството от стойности на даден тип низ са неговите **константи**. Например, “a+b=c-a*e”, “1+3”, “” са константи от тип char[10].

Променлива величина, множеството от допустимите стойности на която съвпада с множеството от стойности на даден тип низ, се нарича променлива от този тип низ. Понякога ще я наричаме само низ.

Фиг. 6.5 определя дефиницията на променлива от тип низ.

Дефиниция на променливи от тип символен низ

<дефиниция_на_променлива_от_тип_низ> ::=
char <променлива>[size] [= “<редица_от_символи>”|
= {<редица_от_константни_изрази>}]$_{опц}$
{,<променлива>[size] [= “<редица_от_символи>”|
={<редица_от_константни_изрази>}]$_{опц}$}$_{опц}$;

където
- <променлива> е идентификатор;
- size е константен израз от интегрален или изброен тип със *положителна* стойност;
- редица от константни изрази се дефинира по следния начин:

<редица_от_константни_изрази> ::= <константен_израз>|
<константен_израз>, <редица_от_константни_изрази>

като константните изрази в случая са от тип char;
- редица от символи се определя по следния начин:

<редица_от_символи> ::= <празно>| <символ> |
<символ><редица_от_символи>

като максималната й дължина е size-1.

Фиг. 6.5 Дефиниция на променливи от тип символен низ

Ще отбележим,че фрагментите:
<променлива>[size]
<променлива>[size] = “<редица_от_символи>” и
<променлива>[size] = {<редица_от_константни_изрази>}
могат да се повтарят. За разделител се използва знакът запетая.
Примери:

```
char s1[5], t[12] = “12345+34”;
char s2[10] = “x+y”, s4, s5 = {‘3’, ‘5’, ‘\0’};
char s3[8] = {‘1’, ‘2’, ‘3’, ‘\0’};
```

При дефиниция на низ с инициализация е възможно size да се пропусне. Тогава инициализацията трябва да съдържа символа ‘\0’ и за стойност на size се подразбира броят на константните изрази, изброени при инициализацията, включително ‘\0’. Ако size е указано, изброените

константни изрази в инициализацията може да са по-малко от size. Тогава останалите се инициализират с ‘\0’.

Примери:

Дефиницията

```
char q[5] =  {‘a’, ‘b’};
```

е еквивалентна на

```
char q[5] =  {‘a’, ‘b’, ‘\0’, ‘\0’, ‘\0’};
```

а също и на

```
char q[5] = “ab”;
```

а

```
char r[] = {‘a’, ‘b’, ‘\0’}; или
char r[] = “ab”;
```

са еквивалентни на

```
char r[3] = {‘a’, ‘b’, ‘\0’}; или
char r[3] = “ab”;
```

Забележка: Не се допускат конструкции от вида:

```
char q[5];
q = {‘a’, ‘v’, ‘s’}; или
char r[5];
r = “avs”;
```

т.е. **на променлива от тип низ не може да бъде присвоявана константа от тип низ.**

Недопустими са също дефиниции от вида:

```
char q[4] = {‘a’, ‘s’, ‘d’, ‘f’, ‘g’, ‘h’}; или
char q[];
```

Инициализацията е един начин за свързване на променлива от тип низ с конкретна константа от множеството от стойности на този тип низ. Друг начин предоставят индексираните променливи.

Примери:

```
q[0] = ‘a’; q[1] = ‘s’; q[2] = ‘d’;
```

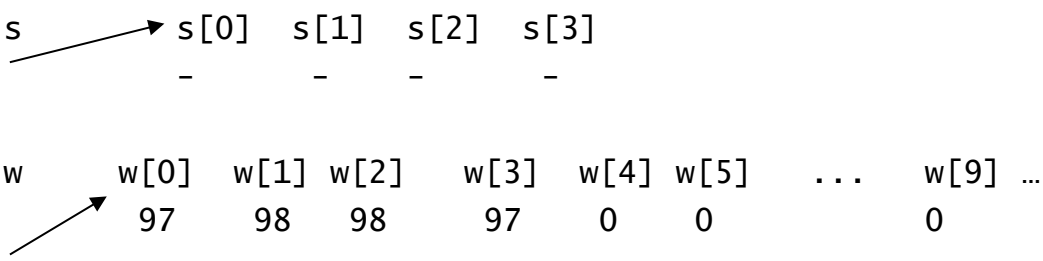
Дефиницията на променлива от тип низ не само свързва променливата с множеството от стойности на указания тип, но и отделя определено количество памет (обикновено 4в), в която записва адреса на първата индексирана променлива, свързана с променливата от тип низ. Останалите индексирани променливи се разполагат последователно след

първата. За всяка индексирана променлива се отделя по 1В ОП. Стойността на отделената за индексираните променливи памет е неопределена освен ако не е зададена дефиниция с инициализация. Тогава в клетките се записват инициализиращите стойности, допълнени със знака за край на низ.

Пример: След дефиницията

```
char s[4];
char w[10] = “abba”;
```

разпределението на паметта има вида:



Операции и вградени функции

Въвеждане на стойност

Реализира се по стандартния начин - чрез оператора cin.

Пример:

```
char s[5], t[3];
cin >> s >> t;
```

Настъпва пауза в очакване да се въведат два низа с дължина **не по-голяма** от 4 в първия и не по-голяма от 2 във втория случай. Водещите интервали, табулации и знакът за преминаване на нов ред се пренебрегват. За разделител на низовете се използват интервалът, табулациите и знакът за преминаване на нов ред. Знакът за край на низ автоматично се добавя в края на всяка от въведените знакови комбинации.

Забележка: При въвеждане на низовете не се извършва проверка за достигане на указаната горна граница. Това може да доведе до трудноотриваеми грешки.

Друг начин за въвеждане на низове дава функцията getline. В тази глава ще я разгледаме непълно и малко неточно. Фиг. 6.6 описва нейните синтаксис и семантика.

Функция getline

Синтаксис

cin.getline(<var_str>, <size>, [<char>]$_{опц}$)

където

- <var_str> е променлива от тип низ;
- <size> е цял израз;
- <char> е произволен символ.

Ако <char> е пропуснато, подразбира се символът \n.

Семантика

Въвежда от буфера на клавиатурата редица от символи с максимална дължина <size>-1. Въвеждането продължава до срещане на символа, зададен в <char> или до въвеждане на <size>-1 символа (ако междувременно не е достигнат символът от <char>).

Фиг. 6.6 Функция getline

Примери:

1.
```
char s1[100];
cin.getline(s1, 10);
```

Настъпва пауза. Очаква се въвеждането на низ, след което да се натисне клавиша ENTER. Ако дължината на въведения низ е по-голяма от 9 преди да е натиснат клавишът ENTER, вземат се първите 9 символа, свързват се с променливата s1 и изпълнението завършва. В противен случай, в s1 се записва редицата от символи без символа \n.

2.
```
char s2[200];
cin.getline(s2, 200, '.');
```

Настъпва пауза. Очаква се въвеждането на низ, което да завърши с натискане на клавиша ENTER. Ако дължината му е по-голяма от 199 преди да е въведен символът '.', вземат се първите 199 символа, свързват се с променливата s2 и изпълнението завършва. В противен случай, в s2 се записва редицата от символи до символа '.' (без него).

Извеждане на низ

Реализира се също по стандартния начин – чрез оператора cout.

Пример:

Операторът

cout << s;

извежда низа, свързан със s. Не е нужно да се грижим за дължината му. Знакът за край на низ идентифицира края на му.

Дължина на низ

Намира се чрез функцията strlen.

Синтаксис

strlen(<str>)

където

<str> е произволен низ.

Семантика

Намира дължината на <str>.

Пример:

strlen(“abc”) намира 3, а strlen(“”) - 0.

За използване на тази функция е необходимо да се включи заглавният файл string.h.

Конкатенация на низове

Реализира се чрез функцията strcat.

Синтаксис

strcat(<var_str>, <str>)

където

- <var_str> е променлива от тип низ;

<str> е низ (константа, променлива или по-общо израз със стойност символен низ).

Семантика

Конкатенира низа от <var_str> с низа <str>. Резултатът от конкатенацията се връща от функцията, а също се съдържа в

променливата <var_str>. За използване на тази функция е необходимо да се включи заглавният файл string.h.

Пример:

```
#include <iostream.h>
#include <string.h>
int main()
{char a[10];
 cout << "a= ";
 cin >> a;     // въвеждане на стойност на a
 char b[4];
 cout << "b= ";
 cin >> b;     // въвеждане на стойност на b
 strcat(a, b);    // конкатениране на a и b, резултатът е в a
 cout << a << '\n';  // извеждане на a
 cout << strlen(strcat(a, b)) << '\n'; //повторна конкатенация
 return 0;
}
```

Забележка: Функцията strcat може да се използва и като оператор, и като израз. Обръщението към strcat от линия 10 е оператор, а това от линия 12 – израз от тип низ.

Сравняване на низове

Реализира се чрез функцията strcmp.

Синтаксис

strcmp(<str1>, <str2>)

където

<str1> и <str2> са низове (константи, променливи или по-общо изрази от тип низ).

Семантика

Низовете <str1> и <str2> се сравняват лексикографски. Функцията strcmp е целочислена. Резултатът от обръщението към нея е цяло число с отрицателна стойност (-1 за реализацията Visual C++ 6.0), ако <str1> е по-малък от <str2>, 0 – ако <str1> е равен на <str2> и с

положителна стойност (1 за реализацията Visual C++ 6.0), ако <str1> е по-голям от <str2>.

За използване на strcmp е необходимо да се включи заглавният файл string.h.

Примери:

1. char a[10] = “qwerty”, b[15] = “qwerty”;
   if (!strcmp(a, b)) cout << “yes \n”;
   else cout << “no \n”;

извежда yes, тъй като strcpm(a, b) връща 0 (низовете са равни), ! strcmp(a, b) е 1 (true).

2. char a[10] = “qwe”, b[15] = “qwerty”;
   if (strcmp(a, b)) cout << “yes \n”;
   else cout << “no \n”;

извежда yes, тъй като strcpm(a, b) връща -1 (низът a е по-малък от b).

3. char a[10] = “qwerty”, b[15] = “qwer”;
   if (strcmp(a, b)) cout << “yes \n”;
   else cout << “no \n”;

извежда yes, тъй като strcpm(a, b) връща 1 (низът a е по-голям от b).

Копиране на низове

Реализира се чрез функцията strcpy.

Синтаксис

strcpy(<var_str>, <str>)

където

- <var_str> е променлива от тип низ;
- <str> е низ (константа, променлива или по-общо израз от тип низ).

Семантика

Копира <str> в <var_str>. Ако <str> е по-дълъг от допустимата за <val_str> дължина, са възможни трудноотриваеми грешки. Резултатът от копирането се връща от функцията, а също се съдържа в <var_str>.

За използване на тази функция е необходимо да се включи заглавният файл string.h.

Пример: Програмният фрагмент

```
char a[10];
strcpy(a, "1234567");
cout << a << "\n";
```

извежда

```
1234567
```

Търсене на низ в друг низ

Реализира се чрез функцията strstr.

Синтаксис

```
strstr(<str1>, <str2>)
```

където

<str1> и <str2> са произволни низове (константи, променливи или по-общо изрази).

Семантика

Търси <str2> в <str1>. Ако <str2> се съдържа в <str1>, strstr връща подниза на <str1> започващ от първото срещане на <str2> до края на <str1>. Ако <str2> не се съдържа в <str1>, strstr връща “нулев указател”. Последното означава, че в позиция на условие, функционалното обръщение ще има стойност false, но при други употреби (например извеждане), ще предизвиква грешка. Програмата ви ще извърши *нарушение при достъп* и ще блокира.

За използване на тази функция е необходимо да се включи заглавният файл string.h.

Примери: Програмният фрагмент

```
char str1[15] = "asemadaemada", str2[10]= "ema";
cout << strstr(str1, str2) << "\n";
```

извежда

```
emadaemada
```

а

```
char str1[15] = "asemadaemada", str2[10]= "ema";
cout << strstr(str2, str1) << "\n";
```

предизвиква съобщение за грешка по време на изпълнение.

Преобразуване на низ в цяло число

Реализира се чрез функцията atoi.

Синтаксис

atoi(<str>)

където <str> е произволен низ (константа, променлива или по-общо израз от тип низ).

Семантика

Преобразува символния низ <str> в число от тип int. Водещите интервали, табулации и знака за преминаване на нов ред се пренебрегват. Символният низ се сканира до първия символ различен от цифра. Ако низът започва със символ различен от цифра и знак, функцията връща 0.

За използване на тази функция е необходимо да се включи заглавният файл stdlib.h.

Примери:

Програмният фрагмент

```
char s[15] = "-123a45";
cout << atoi(s) << "\n";
```

извежда –123, a

```
char s[15] = "b123a45";
cout << atoi(s) << "\n";
```

извежда 0.

Преобразуване на низ в реално число

Реализира се чрез функцията atof.

Синтаксис

atof(<str>)

където <str> е произволен низ (константа, променлива или по-общо израз от тип низ).

Семантика

Преобразува символния низ в число от тип double. Водещите интервали, табулации и знакът за преминаване на нов ред се пренебрегват. Символният низ се сканира до първия символ различен от

цифра. Ако низът започва със символ различен от цифра, знак или точка, функцията връща 0.

За използване на тази функция е необходимо да се включи заглавният файл stdlib.h.

Примери:

Програмният фрагмент

```
char s[15] = "-123.35a45";
cout << atof(st) << "\n";
```

извежда –123.35, a

```
char st[15] = ".123.34c35a45";
cout << atof(st) << "\n";
```

извежда 0.123.

Допълнение:

Конкатенация на n символа от низ с друг низ

Реализира се чрез функцията strncat.

Синтаксис

strncat(<var_str>, <str>, n)

където

- <var_str> е променлива от тип низ;
- <str> е низ (константа, променлива или по-общо израз от тип низ);
- n е цял израз с *неотрицателна* стойност.

Семантика

Копира първите n символа от <str> в края на низа, който е стойност на <var_str>. Копирането завършва когато са прехвърлени n символа, или е достигнат краят на <str>. Резултатът е в променливата <var_str>. За използване на тази функция е необходимо да се включи заглавният файл string.h.

Пример: Резултатът от изпълнението на фрагмента:

```
chat a[10] = “aaaaa”;
strncat(a, “qwertyqwerty”, 5);
```

```
   cout << a;
```
е
```
   aaaaaqwert
```
а на
```
   strncat(a, “qwertyqwerty”, -5);
   cout << a;
```
предизвиква съобщение за грешка (n е отрицателно).

Копиране на n символа в символен низ

Реализира се чрез функцията strncpy.

Синтаксис

```
strncpy(<var_str>, <str>, n)
```
където

- <var_str> е променлива от тип низ;
- <str> е низ (константа, променлива или по-общо израз);
- n е цял израз с *неотрицателна* стойност.

Семантика

Копира първите n символа на <str> в <var_str>. Ако <str> има по-малко от n символа, ‘\0’ се копира до тогава докато не се запишат n символа. Параметърът <var_str> трябва да е от вида char[m], m ≥ n, и съдържа резултатния низ. За използване на тази функция е необходимо да се включи заглавният файл string.h.

Примери: 1. Програмният фрагмент

```
   char a[10];
   strncpy(a, "1234567", 8);
   cout << a << "\n";
```
извежда
```
   1234567
```
Изпълнява се по следния начин: тъй като дължината на низа “1234567” е по-малка от 8, допълва се с един знак ‘\0’ и се свързва с променливата a.

2. Програмният фрагмент

```
   char a[5];
   strncpy(a, "123456789", 5);
   cout << a << "\n";
```

извежда

12345<неопределено>

Изпълнява се по следния начин: тъй като дължината на низа “123456789” е по-голяма от 5, низът “12345” се свързва с променливата a, но не става допълване с ‘\0’, което личи по резултата.

Сравняване на n символа на низове

Реализира се чрез функцията strncmp.

Синтаксис

strncmp(<str1>, <str2>, n)

където

- <str1> и <str2> са низове (константи, променливи или по-общо изрази от тип ния);
- n е цял израз с *неотрицателна* стойност.

Семантика

Сравняват се лексикографски първите n символа на <str1> с низа <str2>. (Ако n е по-голямо от дължината на <str>, сравняват се лексикографски <str1> и <str2>). Резултатът от обръщението към функцията strncmp е цяло число с отрицателна стойност (-1 за Visual C++ 6.0), ако низът от първите n символа на <str1> е по-малък от <str2>, 0 – ако низът от първите n символа на <str1> е равен на <str2> и с положителна стойност (1 за Visual C++ 6.0), ако низът от първите n символа на <str1> е по-голям от <str2>.

За използване на strncmp е необходимо да се включи заглавният файл string.h.

Примери:

1. char a[10] = “qwer”, b[15] = “qwerty”;
   if (!strncmp(a, b, 3)) cout << “yes \n”;
   else cout << “no \n”;

извежда yes, тъй като strncpm(a, b) връща 0 (низовете са равни), ! strncmp(a, b) е 1 (true).

2. char a[10] = “qwer”, b[15] = “qwerty”;
   if (strncmp(a, b, 5)) cout << “yes \n”;
   else cout << “no \n”;

извежда yes, тъй като strncpm(a, b) връща -1 (низът a е по-малък от b).

3. char a[10] = “qwerty”, b[15] = “qwer”;
   if (strncmp(a, b, 5)) cout << “yes \n”;
   else cout << “no \n”;

извежда yes, тъй като strncpm(a, b) връща 1 (низът от първите 5 символа на a е по-голям от b).

Търсене на символ в низ

Реализира се чрез функцията strchr, съдържаща се в string.h.

Синтаксис

strchr(<str>, <expr>)

където

- <str> е произволен низ;
- <expr> е израз от интегрален или изброен тип с положителна стойност, означаваща ASCII код на символ.

Семантика

Търси първото срещане на символа, чийто ASCII код е равен на стойността на <expr>. Ако символът се среща, функцията връща подниза на <str> започващ от първото срещане на символа и продължаващ до края му. Ако символът не се среща - връща “нулев указател”. Последното означава, че в позиция на условие, функционалното обръщение ще има стойност false, но при други употреби (например извеждане), ще предизвиква грешка. Програмата ви ще извърши *нарушение при достъп* и ще блокира.

Примери: Операторът

cout << strchr(“qwerty”, ‘e’);

извежда

erty

Операторът

cout << strchr(“qwerty”, ‘p’);

предизвиква съобщение за грешка, а

if (strchr(“qwerty”, ‘p’)) cout << “yes \n”;
else cout << “no \n”;

извежда no, тъй като ‘p’ не се среща в низа “qwerty”.

Търсене на първата разлика

Реализира се чрез функцията strspn.

Синтаксис

strspn(<str1>, <str2>)

където <str1> и <str2> са произволни низове (константи, променливи или по-общо изрази от тип низ).

Семантика

Проверява до коя позиция <str1> и <str2> съвпадат. Връща дължината на низа до първия различен символ. За използване на тази функция е необходимо да се включи заглавният файл string.h.

Пример: Програмният фрагмент

```
char a[10]= "asdndf", b[15] = "asdsdfdhf";
cout << strspn(a, b) << "\n";
```

извежда 3 тъй като първият символ, по който се различават a и b е в позиция 4.

Задачи върху тип низ

Задача 61. Да се напише програма, която проверява дали низ с дължина не по-голяма от 19 е палиндром, т.е. четен отляво надясно и отдясно наляво е един и същ.

Програма Zad61.cpp решава задачата. Тя обхожда в обратен ред символите на дадения низ a и ги записва в нов низ b (Ще отбележим, че присвояването b[n] = '\0'; е задължително). След това сравнява дадения с новия низ.

```
// Program Zad61.cpp
#include <iostream.h>
#include <string.h>
int main()
{char a[20], b[20];
 cout << "a= ";
```

```
 cin >> a;
 int n = strlen(a);
 int i;
 for (i = n-1; i >= 0; i--)
   b[n-i-1] = a[i];
 b[n] = '\0';
 if (!strcmp(a, b)) cout << "palindrome\n";
 else cout << "not palindrome \n";
 return 0;
}
```

Задача 62. Дадена е редица от n низа с дължина не по-голяма от 9. Да се напише програма, която конкатенира елементите на редицата.

Програма Zad62.cpp решава задачата.

```
// Program Zad62.cpp
#include <iostream.h>
#include <string.h>
int main()
{char a[20][10];  // a е масив от 20 низа char[10]
 cout << "n= ";
 int n;
 cin >> n;
 int i;
 for (i = 0; i <= n-1; i++)
 {cout << "a[" << i << "]= ";
  cin >> a[i];
 }
 char s[200] = ""; // s ще съдържа резултата от конкатенацията
 for (i = 0; i <= n-1; i++)
    strcat(s, a[i]);
 cout << s << '\n';
 return 0;
}
```

Задача 63. Дадена е редица от n низа (n > 1) с дължина не по-голяма от 9. Да се напише програма, която проверява дали редицата е монотонно намаляваща.

Програма Zad63.cpp решава задачата.

```
// Program Zad63.cpp
#include <iostream.h>
#include <string.h>
int main()
{char a[20][10];
 cout << "n= ";
 int n;
 cin >> n;
 int i;
 for (i = 0; i <= n-1; i++)
 {cout << "a[" << i << "]= ";
  cin >> a[i];
 }
 i = -1; int p;
 do
 {i++;
  p = strcmp(a[i], a[i+1]);
 } while ((p == 0 || p == 1) && i < n-2);
 if (p == 0 || p == 1) cout << "yes\n";
 else cout << "no\n";
 return 0;
}
```

Задача 64. Да се напише програма, която сортира във възходящ ред елементите на редица от низове, представящи имена не по-дълги от 9 знака. Сортираната редица да се изведе по 5 думи на ред.

Програма Zad64.cpp решава задачата.

```
// Program Zad64.cpp
#include <iostream.h>
#include <iomanip.h>
```

```
#include <string.h>
int main()
{char a[100][10];
 cout << "n = ";
 int n;
 cin >> n;
 if (!cin)
 {cout << "Error! \n";
  return 0;
 }
 int i;
 for (i = 0; i <= n-1; i++)
 {cout << "a[" << i << "]= ";
  cin >> a[i];
 }
 for (i = 0; i <= n-2; i++)
 {char min[10];
  strcpy(min, a[i]);
  int k = i;
  for (int j = i+1; j <= n-1; j++)
    if (strcmp(a[j], min) == -1)
    {strcpy(min, a[j]);
     k = j;
    }
  strcpy(min, a[i]);
  strcpy(a[i], a[k]);
  strcpy(a[k], min);
 }
 for (i = 0; i <= n-1; i++)
 {cout << setw(10) << a[i];
  if (i != 0 && i % 5 == 0) cout << '\n';
 }
 cout << '\n';
 return 0;
}
```

Задача 65. Дадена е редица от n низа, представящи цели числа. Да се напише програма, която намира средно аритметичното на елементите на редицата.

Програма Zad65.cpp решава задачата.

```
// Program Zad65.cpp
#include <iostream.h>
#include <stdlib.h>
int main()
{char a[20][10];
 cout << "n= ";
 int n;
 cin >> n;
 for(int i = 0; i <= n-1; i++)
 {cout << "a[" << i << "]= ";
  cin >> a[i];
 }
 double average = 0;
 for (i = 0; i <= n-1; i++)
    average = average + atoi(a[i]);
 cout << average/n << '\n';
 return 0;
}
```

Задача 66. Да се напише програма, която въвежда квадратна матрица от низове с дължина не по-голяма от 9. Програмата да проверява дали матрицата е симетрична относно главния си диагонал.

Програма Zad66.cpp решава задачата.

```
// Program Zad66.cpp
#include <iostream.h>
#include <string.h>
int main()
{char a[20][20][10];
 cout << "n= ";
```

```
    int n;
    cin >> n;
    int i, j;
    for (i = 0; i <= n-1; i++)
       for (j = 0; j <= n-1; j++)
       {cout << "a[" << i << "][" << j << "]= ";
        cin >> a[i][j];
       }
    i = 0; int p;
    do
    {i++;
     j = -1;
     do
     {j++;
      p = !strcmp(a[i][j], a[j][i]);
     } while (p && j < i-1);
    } while (p && i < n-1);
    if (p) cout << "yes\n";
    else cout << "no \n";
    return 0;
   }
```

Забележка: В условията на циклите do/while е използван аритметичен израз на мястото на предикат (функцията strcmp е целочислена).

Задачи

Задача 1. Да се напише програма, която намира скаларното произведение на реалните вектори a = (a_0, a_1, ..., a_{n-1}) и b = (b_0, b_1,..., b_{n-1}), ($1 \le n \le 50$).

Задача 2. Да се напише програма, която въвежда n символа и намира и извежда минималния (максималния) от тях.

Задача 3. Да се напише програма, която:

а) въвежда редицата от n цели числа a_0, a_1, ..., a_{n-1};

б) намира и извежда сумата на тези елементи на редицата, които се явяват удвоени нечетни числа.

Задача 4. Да се напише програма, която намира и извежда сумата от положителните и броя на отрицателните елементи на редицата от реални числа a_0, a_1, ..., a_{n-1} ($1 \le n \le 30$).

Задача 5. Да се напише програма, която изчислява реципрочното число на произведението на тези елементи на редицата a_0, a_1, ..., a_{n-1} ($1 \le n \le 50$), за които е в сила релацията $2 < a_i < i!$.

Задача 6. Да се напише програма, която изяснява, има ли в редицата от цели числа a_0, a_1, ..., a_{n-1} ($1 \le n \le 100$) два последователни нулеви елемента.

Задача 7. Дадени са сортираните във възходящ ред числови редици a_0, a_1, ..., a_{k-1} и b_0, b_1, ..., b_{k-1} ($1 \le k \le 40$). Да се напише програма, която намира броя на равенствата от вида $a_i = b_j$ ($i = 0, ..., k-1, j = 0, ..., k-1$).

Задача 8. Дадени са две редици от числа. Да се напише програма, която определя колко пъти първата редица се съдържа във втората.

Задача 9. Всяка редица от равни числа в едномерен сортиран масив, се нарича площадка. Да се напише програма, която намира началото и дължината на най-дългата площадка в даден сортиран във възходящ ред едномерен масив.

Задача 10. Да се напише програма, която по дадена числова редица a_0, a_1, ..., a_{n-1} ($1 \le n \le 20$) намира дължината на максималната й ненамаляваща подредица a_{i1}, a_{i2}, ..., a_{ik} ($a_{i1} <= a_{i2} <= ... <= a_{ik}$, $i_1 < i_2 < ... < i_k$).

Задача 11. Дадена е редицата от символи s_0, s_1, ..., s_{n-1} ($1 \le n \le 15$). Да се напише програма, която извежда отначало всички символи, които изобразяват цифри, след това всички символи, които изобразяват малки латински букви и накрая всички останали символи от редицата, запазвайки реда им в редицата.

Задача 12. Дадени са две цели числа, представени с масиви от символи. Да се напише програма, която установява дали първото от двете числа е по-голямо от второто.

Задача 13. Да се напише програма, която определя дали редицата от символи s_0, s_1, ..., s_{n-1} ($1 \le n \le 45$) е симетрична, т.е. четена отляво надясно и отдясно наляво е една и съща.

Задача 14. Дадена е редицата от естествени числа a_0, a_1, ..., a_{n-1} ($1 \le n \le 30$). Да се напише програма, която установява има ли сред

елементите на редицата не по-малко от два елемента, които са степени на 2.

Задача 15. Дадени са полиномите Pn(x) и Qm(x). Да се напише програма, която намира:

а) сумата им;

б) произведението им.

Задача 16. За векторите $a = (a_0, a_1, \ldots, a_{n-1})$ и $b = (b_0, b_1, \ldots, b_{n-1})$ $(1 \le n \le 20)$, да се определи дали са линейно зависими.

Задача 17. Дадена е квадратна целочислена матрица A с размерност nxn, елементите на която са естествени числа. Да се напише програма, която намира:

а) сумата от елементите под главния диагонал, които са прости числа;

б) произведението от елементите над главния диагонал, в записа на цифрите на които се среща цифрата 5;

в) номера на първия неотрицателен елемент върху главния диагонал.

Задача 18. Дадена е квадратната реална матрица A от n-ти ред. Да се напише програма, която намира:

а) сумата от елементите върху вторичния главен диагонал;

б) произведението от елементите под (над) вторичния главен диагонал.

Задача 19. Дадена е реална правоъгълна матрица A с размерност nxm. Да се напише програма, която изтрива k-тия ред (стълб) на A. Изтриването означава да се преместят редовете (стълбовете) с един нагоре (наляво) и намаляване броя на редовете (стълбовете) с един.

Задача 20. Върху равнина са дадени n точки чрез матрицата

$$x = \begin{pmatrix} x_{0,0} & x_{0,1} & \ldots & x_{0,n-1} \\ x_{1,0} & x_{1,1} & \ldots & x_{1,n-1} \end{pmatrix}$$

така, че $(x_{0,i}, x_{1,i})$ са координатите на i-тата точка. Точките по двойки са съединени с отсечки. Да се напише програма, която намира дължината на най-дългата отсечка.

Задача 21. Дадена е матрицата от цели числа

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & \ldots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \ldots & a_{1,n-1} \end{pmatrix}$$

Да се напише програма, която намира сумата на тези елементи $a_{1,i}$, ($0 \le i \le n-1$), за които $a_{0,i}$ имат стойността на най-голямото сред елементите от първия ред на матрицата.

Задача 22. Дадено е множеството M от двойки

$$M = \{<x_0, y_0>, <x_1, y_1>, \ldots, <x_{n-1}, y_{n-1}>\},$$

като x_i и y_i ($0 \le i \le n-1$) са цели числа. Да се напише програма, която проверява дали множеството M дефинира функция.

Упътване: Множеството M дефинира функция, ако от $x_i = x_j$ следва $y_i = y_j$.

Задача 23. Дадена е целочислената квадратна матрица A от n-ти ред. Да се напише програма, която намира максималното от простите числа на A.

Задача 24. Казваме, че два стълба на една матрица си приличат, ако съвпадат множествата от числата, съставящи стълбовете. Да се напише програма, която намира номерата на всички стълбове на матрицата Anxm, които си приличат.

Задача 25. Матрицата A има седлова точка в $a_{i,j}$, ако $a_{i,j}$ е минимален елемент в i-я ред и максимален елемент в j-я стълб на A. Да се напише програма, която намира *всички* седлови точки на дадена матрица A.

Задача 26. Матрицата A има седлова точка в $a_{i,j}$, ако $a_{i,j}$ е минимален елемент в i-я ред и максимален елемент в j-я стълб на A. Да се напише програма, която установява дали *съществува* седлова точка в дадена матрица A.

Задача 27. Дадена е квадратна матрица A от n-ти ред. Да се напише програма, която установява дали съществува k ($0 \le k \le n-1$), така че k-я стълб на A (обхождан отгоре надолу) да съвпада с k-я й ред (обхождан отляво надясно).

Задача 28. Дадена е реалната квадратна матрица Anxn. Да се напише програма, която намира:

а) $\max\limits_{0\le i\le n-1} \{ \min\limits_{0\le j\le n-1} \{aij\}\}$

б) $\max\limits_{0\le j\le n-1} \{ \min\limits_{0\le i\le n-1} \{aij\}\}$

в) $\min\limits_{0\le i\le n-1} \{ \max\limits_{0\le j\le n-1} \{aij\}\}$

г) $\min\limits_{0\le j\le n-1} \{ \max\limits_{0\le i\le n-1} \{aij\}\}$

Задача 29. Дадена е квадратната матрица Anxn ($2 \le n \le 10$). Да се напише програма, която определя явява ли се A ортонормирана, т.е.

такава, че скаларното произведение на всеки два различни реда на A е равно на 0, а скаларното произведение на всеки ред със себе си е равно на 1?

Задача 30. Да се напише програма, която определя явява ли се квадратната матрица Anxn магически квадрат, т.е. такава, че сумата от елементите от всички редове и стълбове е еднаква.

Задача 31. Дадена е система от линейни уравнения от n-ти ред. Да се напише програма, която я решава.

Задача 32. С тройката (i, j, v) се представя елемента v от i-я ред и j-я стълб на матрица. Две матрици с размерност nxn са представени като редици от тройки. Тройките са подредени по редове. Ако тройката (i, j, v) отсъства, приема се, че v = 0. Да се напише програма, която събира матриците и представя резултата като редица от тройки.

Задача 33. Да се напише програма, която сортира във възходящ ред елементите на всеки от редовете на квадратна матрица от низове, изразяващи думи с максимална дължина 9.

Задача 34. Дадена е квадратна матрица Anxn от низове, представящи думи с максимална дължина 9. Да се напише програма, която намира броя на палиндромите в частта над главния диагонал на A.

Задача 35. Дадена е квадратна матрица Anxn от низове, представящи думи с максимална дължина 6. Да се напише програма, която намира колко пъти думата s се среща в частта под вторичния главен диагонал на A.

Задача 36. Дадена е квадратна матрица Anxn от низове, представящи думи с максимална дължина 6. Да се напише програма, която проверява дали думата s се среща в частта над вторичния главен диагонал на A.

Задача 37. Дадена е квадратна матрица Anxn от низове, съдържащи думи с максимална дължина 9. Да се напише програма, която проверява дали изречението, получено след конкатенацията на думите от главния диагонал съвпада с изречението, получено по аналогичен начин за думите от вторичния главен диагонал на A.

Задача 38. Дадена е квадратна матрица A от n-ти ред от низове, съдържащи думи с максимална дължина 9. Да се напише програма, която намира изречението, получено след обхождане на A по спирала, започвайки от горния ляв ъгъл.

Допълнителна литература

1. B. Stroustrup, C++ Programming Language. Third Edition, Addison – Wesley, 1997.

2. К. Хорстман, Принципи на програмирането със C++, С., СОФТЕХ, 2000.

3. М. Тодорова, Програмиране на Паскал, Полипринт, София, 1993.

4. Ст. Липман, Езикът C++ в примери, “КОЛХИДА ТРЕЙД” КООП, София, 1993.