Увод в програмирането

Съставни типове данни.

Масив. Символен низ

Структури от данни

- 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.

Структури от данни

- 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**.

Структура от данни масив

- **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.

Структура от данни масив …

- **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

Тип масив

- В C++ структурата от данни масив се 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.

Тип масив …

- **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

  size - константен израз от интегрален или изброим тип с положителна стойност

  Тогава **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

Тип масив …

- *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.

Тип масив …

- **Множество от стойности**

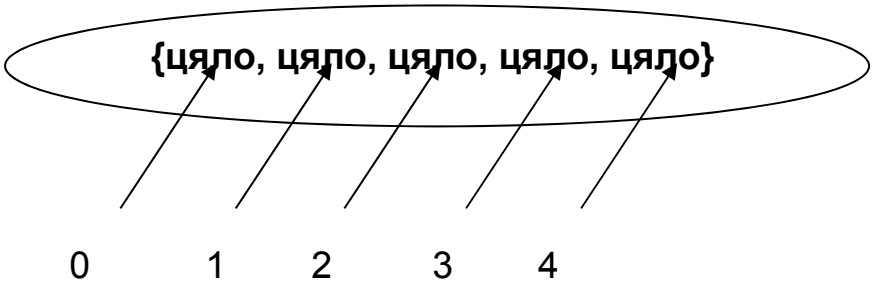
  Множеството от стойности на типа T[size] се състои от всички редици от по size елемента, които са произволни константи от тип T.

  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

Тип масив …

- *Примери:*

  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.



Тип масив …

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

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

  {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].

Тип масив …

- 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**.

Тип масив …

- **Дефиниция на масив**
  <дефиниция_на_променлива_от_тип_масив> ::=
  T <променлива>[size] [= {<редица_от_константни_изрази>}]
  {,<променлива>[size] [= {<редица_от_константни_изрази>}] };
  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.

Тип 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

  T <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

Тип масив …

- *Примери:*

```
int q[5] = {1, 2, 3};
```
е еквивалентна на
```
int q[] =  {1, 2, 3, 0, 0};

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};

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

Тип масив …

- Фрагментите
  <променлива>[size] и
  <променлива>[size] = {<редица_от_константни_изрази>}
  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];
  ```

Тип масив …

- 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.
- **Синтаксис на индексирани променливи**

  <индексирана_променлива> ::=

  <променлива_от_тип_масив>[<индекс>]

  където
  - <индекс> е израз от интегрален или изброен тип.
  - всяка индексирана променлива е от базовия тип.

Тип масив …

- *Примери:*

1. `int a[5];`
   С променливата a са свързани индексираните променливи a[0], a[1], a[2], a[3] и a[4], които са от тип int.
2. `double b[10];`
   С променливата b са свързани индексираните променливи b[0], b[1],…, b[9], които са от тип double.
3. `enum {FALSE, TRUE} x[20];`
   С променливата x са свързани индексираните променливи x[0], x[1],…, x[19], които са от тип enum {FALSE, TRUE}.

Тип масив …

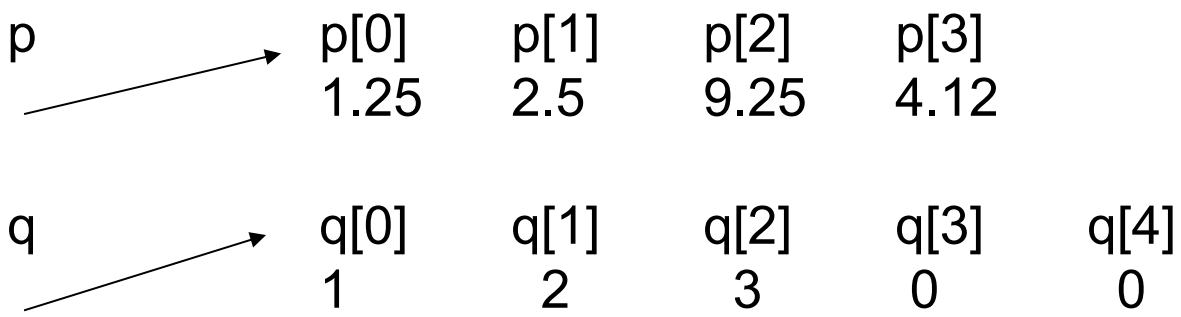
- Дефиницията на променлива от тип масив не само свързва променливата с множеството от стойности на указания тип, но и отделя определено количество памет (обикновено 4B), в която записва адреса в паметта на първата индексирана променлива на масива. Останалите индексирани променливи се разполагат последователно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

ОП				
a	a[0]	a[1]	…	a[4]
адрес на a[0]	-	-	…	-
4B	4B	4B	…	4B

Тип масив …

- За краткост, вместо “адрес на a[0]” ще записваме стрелка от a към a[0].
- Пример:

```
double p[4] = {1.25, 2.5, 9.25, 4.12};
int q[5] = {1, 2, 3};
```



Тип масив …

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

```
#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;
   }
```

Задачи върху тип масив …

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

Задачи върху тип масив …

```
   // извеждане на елементите
   // на x в обратен ред
   for (i = n-1; i >= 0; i--)
       cout << x[i] << "\n";
   return 0;
}
```

Задачи върху тип масив …

- *Забележка:* Фрагментите:

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

и

```
…
int n = 10;
int x[n];
…
```

са недопустими, тъй като n не е константен израз.

- Фрагментът

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

е допустим.

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.

Приложения на структурата от данни масив …

- **Задача.** 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

```
int a[20];
cout << "n= ";
int n;
cin >> n; // въвеждане на дължината на редицата
…
 // въвеждане на редицата
…
 // въвеждане на стойност за x
 int x;
 cout << "x= ";
 …
 // намиране на броя br на срещанията на x в редицата
 int br = 0;
 for (int i = 0; i <= n-1; i++)
   if (a[i] == x) br++;
 cout << "number = " << br << "\n";
```

Приложения на структурата от данни масив …

- **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 $a_0 \leq a_1 \leq ... \leq a_{n-1}$ или $a_0 \geq a_1 \geq ... \geq a_{n-1}$. 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.
  - Метод на пряката селекция
  - Метод на мехурчетата
  - Quick Sort

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.

Многомерни масиви …

- *Примери:*

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

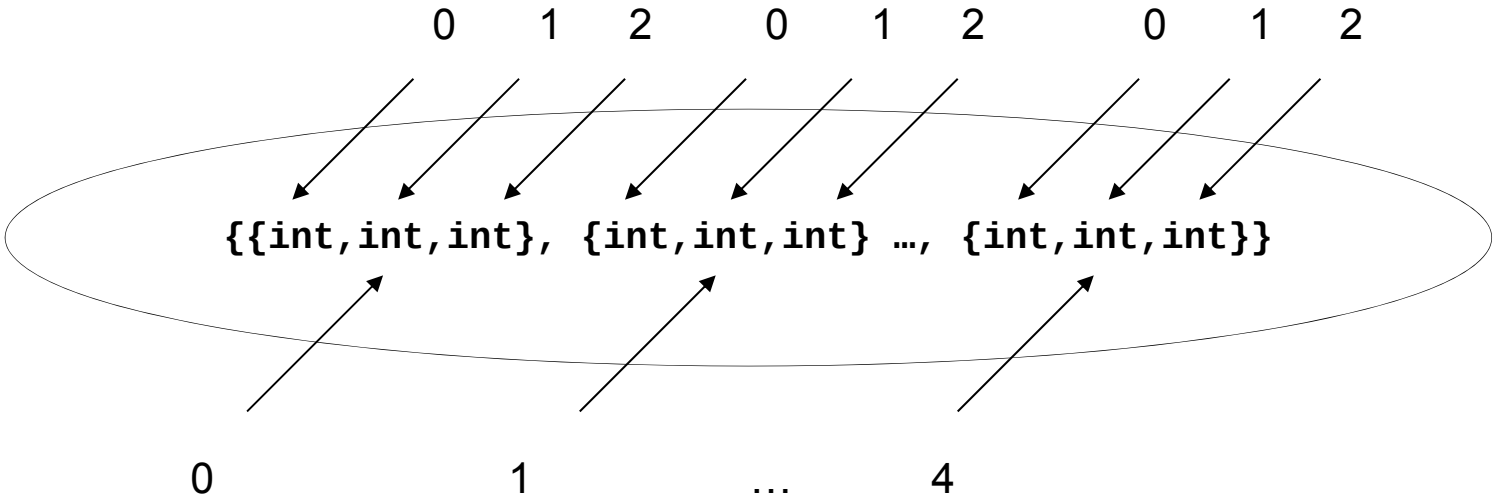
  `double [4][5][3]`
  дефинира тримерен масив от тип double;
- **Множество от стойности**

  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 от 0 до size$_1$-1. Елементите от множеството от стойности на даден тип многомерен масив са **константите** на този тип масив.

Многомерни масиви …

- *Пример:*

  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.



Многомерни масиви …

- 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**.

Многомерни масиви …

- **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>}]
{,<променлива>[size$_1$][size$_2$]…[size$_n$]
[= {<редица_от_константи_от_тип T1>}]};

- където
  - T е име или дефиниция на произволен тип, различен от псевдоним, void и функционален;
  - T1 е име на типа T[size2]…[sizen];
  - 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.

Многомерни масиви …

- *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}} };
```

Многомерни масиви …

- 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.
- **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.

Многомерни масиви …

- *Примери:*

  ```
  int x[10][20];
  ```

  С променливата x 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],
```

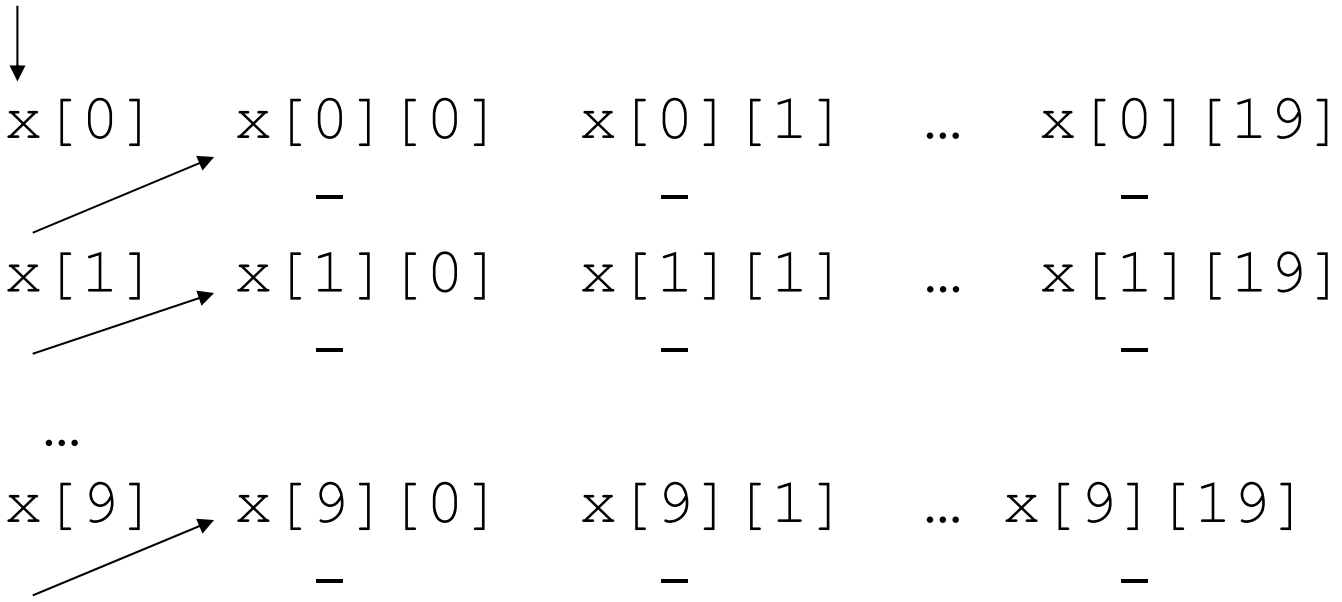
които са от тип int.

Многомерни масиви …

- Дефиницията на променлива от тип многомерен масив отделя и 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.

Многомерни масиви …

```
int x[10][20];
```



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.

```
#include <iostream.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;
   }
```

Многомерни масиви …

```
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;
}
```

Многомерни масиви …

```
 // 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;
```

Многомерни масиви …

```
  // извеждане на матрицата b по редове
  for (i = 0; i <= n-1; i++)
  {   for (j = 0; j <= m-1; j++)
          cout << b[i][j] << ' ';
      cout << '\n';
  }
  return 0;
}
```

Символни низове
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 a “ е подниз на низа “This is a string.”, а низът “ is a sing” не е негов подниз.

Символни низове
Структура от данни низ …

- **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 изчерпан, приема се, че първият низ е по-малък от втория. Ако пък е по-голям или вторият низ е изчерпан – приема се, че първият низ е по-голям от втория. Ако в процеса на сравнение и двата низа едновременно са изчерпани, те са равни. Това сравнение се нарича **лексикографско**.

Символни низове
Структура от данни низ …

- *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.

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.

  Те са семантично еквивалентни.
- В тази част ще разгледаме символните низове като масиви от символи.
- Примери:

```
char str1[100];
char str2[5] = {'a', 'b', 'c'};
char str2[5] = {'a', 'b', 'c', '\0', '\0'};
char str2[5] = {'a', 'b', 'c', 0, 0};
```

Символни низове в езика C++

- 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
  ```

- 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.

Символни низове в езика C++

- *Примери:* Дефинициите

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

- 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.

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].

Тип символен низ …

- 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

Тип символен низ …

- **Дефиниция на променливи от тип символен низ**

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

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.

Тип символен низ …

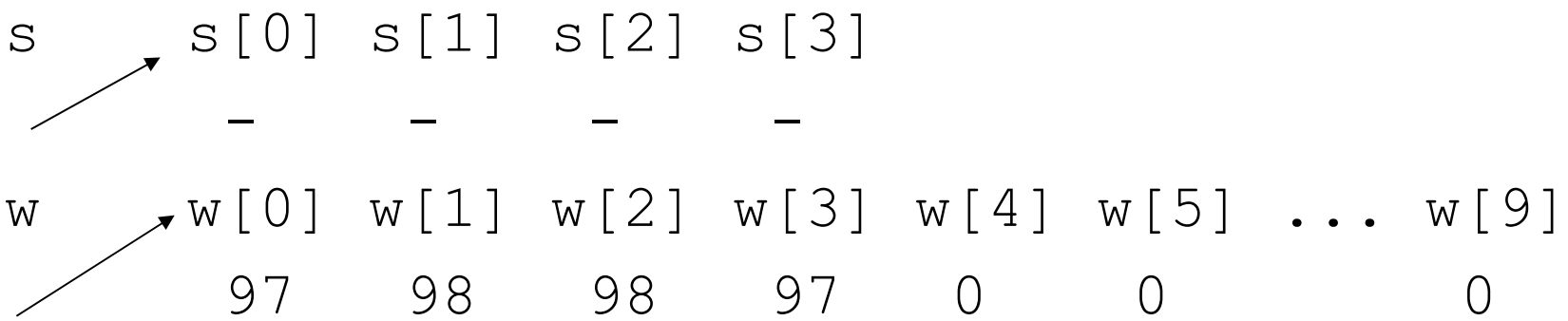
- *Примери:*

```
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’

Тип символен низ …

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



Тип символен низ … Операции и вградени функции

- **Въвеждане на стойност**

  Реализира се по стандартния начин - чрез оператора >>.

- *Пример:*

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

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

Тип символен низ … Операции и вградени функции

- **Функция getline**
- *Синтаксис*

  cin.getline(<var_str>, <size>, [<char>])

  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>).

Тип символен низ … Операции и вградени функции

- *Примери:*

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

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

- **Извеждане на низ**

  Реализира се също по стандартния начин – чрез оператора <<.

```
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.

Тип символен низ … Операции и вградени функции

- **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.

Тип символен низ … Операции и вградени функции

- **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.

Тип символен низ … Операции и вградени функции

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;
}
```

Тип символен низ … Операции и вградени функции

- **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++), ако <str1> е по-малък от <str2>, 0 – ако <str1> е равен на <str2> и с положителна стойност (1 за реализацията Visual C++), ако <str1> е по-голям от <str2>.
  За използване на strcmp е необходимо да се включи заглавният файл string.h.

Тип символен низ … Операции и вградени функции

- *Примери:*

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

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

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

Тип символен низ … Операции и вградени функции

- **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.

Тип символен низ … Операции и вградени функции

- *Пример:*

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

- **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).

Тип символен низ … Операции и вградени функции

- *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.

Тип символен низ … Операции и вградени функции

- *Примери:*

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

извежда emadaemada

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

предизвиква съобщение за грешка по време на изпълнение.

Тип символен низ … Операции и вградени функции

- **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

Тип символен низ … Операции и вградени функции

- *Примери:*

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

извежда –123

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

извежда 0.

Тип символен низ … Операции и вградени функции

- **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.

Тип символен низ … Операции и вградени функции

- *Примери:*

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

извежда –123.35, а

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

извежда 0.123.