Многомерные массивы

int matr[6][8] – массив из 6 строк и восьми столбцов.
Инициализация многомерного массива
int mass1 [ ][ ] = { {1, 1}, {0, 2}, {1, 0} };
int mass2 [3][2] = {1, 1, 0, 2, 1, 0};

Пример. Программа определяет в целочисленной матрице номер строки, которая содержит наибольшее количество элементов, равных нулю. Переменная istr – номер искомой строки, Kol – количество нулевых элементов в текущей (i-ой) строке, MaxKol – максимальное количество нулевых элементов.

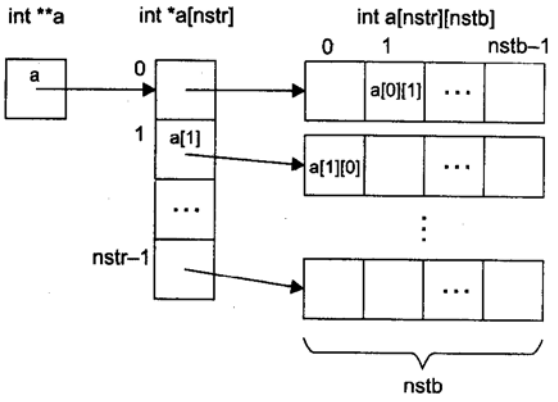
```
#include <stdio.h>
int main(){
   const int nstr = 4, nstb = 5;              // размерности массива
   int b[nstr][nstb];                         // описание массива
   int i, j;
   for (i = 0; i<nstr; i++)                   // ввод массива
   for (j = 0; j<nstb; j++) scanf("%d", &b[i][j]);
   int istr = -1, MaxKol = 0;
   for (i = 0; i<nstr; i++){                  // просмотр массива по строкам
      int Kol = 0;
      for (j = 0; j<nstb; j++) if (b[i][j] == 0)Kol++;
      if (Kol > MaxKol){istr = i; MaxKol = Kol;}
   }
   printf(" Исходный массив:\n");
   for (i = 0; i<nstr; i++){
      for (j = 0; j<nstb; j++)printf("%d ", b[i][j]);
      printf("\n");}
   if (istr == -1)printf("Нулевых элементов нет");
   else printf("Номер строки: %d", istr);
   return 0;}
```

Выделение памяти под многомерный массив – самая левая размерность может быть переменной:

```
int nstr = 5;
int ** m = (int **) new int [nstr][10];
```

Обе размерности задаются динамически на этапе выполнения:

```
int nstr, nstb;
cout << " Введите количество строк и столбцов :";
cin >> nstr >> nstb;
int **a = new int *[nstr];      // 1
for(int i = 0; i<nstr; i++)     // 2
   a[i] = new int [nstb];       // 3
```



Выделение памяти под двумерный массив

Строки

Строка- массив символов заканчивающихся нуль-символом (‘\0’).

```
char str[10] = "Vasia";
// выделено 10 элементов с номерами от 0 до 9
// первые элементы – 'V', 'a', 's', 'i', 'a', '\0'
```

Если строка при определении инициализируется, то ее размерность можно опускать

```
char str[ ] = “Vasia” ; // выделено и заполнено 6 байт.
```

Инициализация строковой константы (последующее изменение не допускается)

```
char *str = “Vasia”;
```

Функции работы со строками стандартной библиотеки языка С

Объявлены в заголовочных файлах <string.h> и <cstring>)
char *__strcat__(char *s1, char *s2); - добавляет строку s2 в конец строки s1;
char *__strchr__(char *s, int ch); - ищет символ в строке;
int __strcmp__(char *s1, char *s2); сравнивает строки и возвращает ноль, если s1 равно s2, отрицательное, если s1 меньше s2, и положительное, если наоборот;
int __strcoll__(char *s1, char *s2); сравнивает строки с учетом локализации setlocale
char *__strcopy__(char *s1, char *s2); - копирует строку s2 в строку s1, возвращает s1;
size_t __strlen__(char *s) – возвращает длину строки;
char *__strstr__(char *s1, char *s2); - ищет первое вхождение подстроки s2 в строке s1, возвращает указатель на место первого вхождения, если поиск неудачен – возвращает NULL;
size_t __strcspn__(char *s1, char *s2); - возвращает длину начальной подстроки строки s1, не содержащей ни одного символа из строки s2, то есть индекс первого вхождения какого-либо символа строки s2 в строке s1;
strncat ( ), strncmp( ), strncopy( ) - учитывают первые n символов строки s2.

Пример. Программа запрашивает пароль не более трех раз.

```
#include <stdio.h>
#include <string.h>
int main(){
   char s[5], passw[] = "kuku";          // passw - эталонный пароль.
                                         // Можно описать как *passw = "kuku";
   int i, k = 0;
   for (i = 0; !k && i<3; i++){
      printf("\nвведите пароль (4 символа):\n");
      gets(s);                           // функция ввода строки
      if (strcmp(s, passw) == 0)k = 1;   // функция сравнения строк
   }
   if (k) printf("\nпароль принят");
   else printf("\nпароль не принят");
   return 0;
}
```

Копирование строки в строку – первый способ

```
char src[10], dest[10];
…
for (int i = 0; i<=strlen(src); i++) dest[i] = src[i]
```

Копирование строки в строку – второй способ

```
#include <iostream.h>
int main(){
   char *src = new char [10];
   char *dest = new char [10], *d = dest;
   cin >> src;
   while ( *src != 0) *d++ = *src++;
   *d = 0;      // завершающий нуль
   cout << dest;
   return 0;
}
```

Можно и так: `while(*d++ = *src++);`

Копирование строки в строку – третий способ

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main( ) {
   char s1[ ]="abcdef", s2[ ]="yes";
        strcpy(s1,s2);
        printf("%s", s1);
}
```

Выводит “yes”

Типы данных, определяемые пользователем

Переименование типов (typedef)

typedef тип новое_имя [размерность];

```
typedef unsigned int UINT;
typedef char Msg[100];
typedef struct{
   char fio[30];
   int date, code;
   double salary;} Worker;
```

Использование новых типов

```
UINT i, j;            // две переменных типа unsigned int
Msg str[10];          // массив из 10 строк по 100 символов
Worker stuff[100];    // массив из 100 структур
```

Перечисления (enum)

enum [имя_типа] {список_констант};

```
enum Err {ERR_READ, ERR_WRITE, ERR_CONVERT};
Err error;
…
switch (error){
      case ERR_READ:     /* операторы */ break;
      case ERR_WRITE:    /* операторы */ break;
      case ERR_CONVERT:  /* операторы */ break;
}
```

Структуры (struct)

Структура –составной тип данных, который может содержать элементы разных типов. В C++ структура – частный случай класса.

```
struct [ имя_типа ] {
   тип_1 элемент_1;
   тип_2 элемент_2;
   …
   тип_n элемент_n;
} [ список_описателей ];
```

```
// Определение массива структур и указателя на структуру:
struct {
   char fio[30];
   int date, code;
   double salary;
}stuff[100], *ps;
```

Имя структуры является новым типом данных, который можно использовать для определения указателей и выделения памяти аналогичным структурам

```
struct Worker{     // описание нового типа Worker
   char fio[30];
   int date, code;
   double salary;
}; // описание заканчивается точкой с запятой
// определение массива типа Worker и указателя на тип Worker:
Worker stuff[100], *ps;
```

Инициализация структуры

```
struct{
   char fio[30];
   int date, code;
   double salary;
}worker = {"Страусенко", 31, 215, 3400.55};
```

Доступ к полям структуры выполняется с помощью оператора выбора «.» - точка при обращении к полю через имя структуры и с помощью «->» - стрелка при обращении через указатель:

```
Worker worker, stuff[100], *ps;
...
worker.fio = "Страусенко";
stuff[8].code = 215;
ps->salary = 0.12;
```

Битовые поля

Особый вид структуры, используемый для хранения плотно упакованных двоичных данных

```
struct Options{
   bool centerX:1;
   bool centerY:1;
   unsigned int shadow:2;
   unsigned int palette:4;
};
```

Объединения (union)

Объединение – частный случай структуры, все поля которой располагаются по одному адресу. Используется для экономии памяти. когда известно, что больше одного поля одновременно не потребуется

```
#include <iostream.h>
int main(){
   enum paytype {CARD, CHECK};
   paytype ptype;
   union payment{
      char card[25];
      long check;
   } info;
    /* присваивание значений info и ptype */
   switch (ptype){
      case CARD: cout << "Оплата по карте: " << info.card; break;
      case CHECK: cout << "Оплата чеком: " << info.check; break;
   }
   return 0;
}
```

Объединения часто используются для разной интерпретации одного того же битового представления

```
struct Options{
   bool centerX:1;
   bool centerY:1;
   unsigned int shadow:2;
   unsigned int palette:4;
};
   union{
      unsigned char ch;
      Options bit;
   }option = {0xC4};
   cout << option.bit.palette;
   option.ch &= 0xF0; // наложение маски
```