**Структурированные типы данных**

1. **Массивы.**
2. **Структуры.**
3. **Перечисления.**
4. **Смеси.**
5. **Массивы**

**Определение:** Массивы — логически связанные группы элементов одинакового типа.

Ко всему массиву целиком можно обращаться по имени. Также можно выбирать любой элемент массива. Число элементов массива (размер массива) задается при его объявлении и в дальнейшем не меняется (массив в Си является статическим).

Элементы массива нумеруются (индексируются) от 0 до размер массива-1.

Массивы определяются также как и переменные.

**Пример:**

int a[100]; // массив из 100 элементов целого типа a[0], …,a[99]

char b[30]; // 30 элементов типа char

float c[42]; // 42 элемента типа float

**Инициализация массивов**

Начальное значение массиву можно присвоить, указав список значений.

**Пример:**

int iVect[] = { 1,2,3,4 };

char cStr[] = { 'm','a','p','\0' };

char cStr1[] = "map";

float fVect[3] = { 0.5, 1.7, 3.8 };

соответственно **4, 3** и **3**. В четвертом случае размерность массива указана явно и равна **3**.

**Пример**: // Проверка

#include<stdio.h>

#include<stdlib.h>

int main()

{

char cStr[] = { 'm','a','p','\0' };

char cStr1[] = "map";

printf("cStr=%d\tcStr1=%d\n", sizeof(cStr), sizeof(cStr1));

system("pause");

return 0;

}

Если размер явно указан, то задание большего числа элементов в списке инициализации будет ошибкой.

Если в списке элементов инициализации недостает элементов, всем остальным элементам массива присваивается значение **0**.

**Пример:**

int iVect1[3]={1,2,3,4}; // ошибка

int iVect2[5]={1,2,3} ; // равнозначно {1,2,3,0,0}

**Замечание:** Такой вид присваивания массиву ошибочен

**int iVect2[5];**

**. . .**

**iVect2={1,2,3,4,5};**

**Многомерные массивы**

Двухмерный массив представляется как одномерный массив, элементы которого являются тоже массивами, то есть как массив массивов.

**Пример**:

char cArr[10][20];

Элементы двухмерного массива хранятся по строкам (то есть быстрее всего изменяется крайний правый индекс).

**a[5][9]** — десятый элемент шестой строки.

В языке Си существует сильная взаимосвязь между указателями и массивами.

Пусть дано:

**int a[5];** // массив из 5 элементов типа int.

**int \*y;** // y – указатель на тип int.

Тогда:

Так как имя массива есть адрес его нулевого элемента, то оператор **y=&a[0];** эквивалентен **y=a;**

Оператор

**y=&a[2];** присваивает переменной **y** адрес третьего элемента массива.

Если указатель **y** указывает на очередной элемент массива **a**, то **y=y+1;** указывает на следующий элемент массива.

В нашем примере **y** указывает теперь на элемент **a[3]**.

Соответственно, через оператор **\*** и указатель **y**, можно получить значение элемента массива.

Так, следующие два оператора:

**int c=a[3];**

**int c=\*y;**

эквивалентны.

Пусть **y=&a[0];** //(**y=a;**)

Элемент

**a[i]** можно представить как **\*(a+i),** или **\*(y+i),** или **y[i].**

Таким образом, любой массив и индексное выражение в массиве можно представить посредством указателей.

**Замечание:** Есть различие между указателем и именем массива.

**Указатель** — это переменная и **y=a;** или **y++**; – допустимые операции.

**Имя массива** — константа. Выражения вида **a=y; a++;** – использовать нельзя, так как a – константа и не может быть изменена.

**Замечание:**  Если **а** адрес, то нельзя использовать оператор

**y=&a;**

**(**так как **a** – адрес).

/\* Если указатели адресуют элементы одного массива, то их можно сравнивать **(>, <, ==, !=).**

Нельзя сравнивать, либо применять в арифметических операциях указатели на разные массивы. (**NULL** вместо нуля). \*/

Указатели на элементы одного массива также можно вычитать. Тогда результатом будет число элементов массива, расположенных между уменьшаемым и вычитаемым объектами.

#include<stdio.h>

#include <windows.h>

#include<stdlib.h>

int main()

{

int iArr[10];

int \*iPtr1, \*iPtr2, iVar;

char cStr[] = "Введите элементы массива ";

CharToOem(cStr, cStr);

printf("%s\n", cStr);

for (int i = 0; i < 10; i++)

scanf\_s("%d", &iArr[i]);

for (int i = 0; i < 10; i++)

printf("iArr[%d]=%2d\n", i, iArr[i]);

// Работа с указателями

iPtr1 = iArr;

iPtr2 = &iArr[0];

printf("iArr=%p\tiPtr1=%p\tiPtr2=%p\n", iArr, iPtr1, iPtr2);

printf("iArr[2]=%d\t\*(iArr+2)=%d\n", iArr[2], \*(iArr + 2));

printf("iPtr1[2]=%d\t\*(iPtr1+2)=%d\n", iPtr1[2], \*(iPtr1 + 2));

iPtr1 = &iArr[2];

iPtr2 = &iArr[6];

iVar = iPtr2 - iPtr1;

printf("iVar=%d\n", iVar);

// Операторы с ошибками

// iArr++;

// iArr=iPtr1;

// iPtr1=&iArr;

system("pause");

return 0;

}

**Определение массива с инициализацией:**

**Пример**:// Инициализация массивов

#include<stdio.h>

#include <windows.h>

#include<stdlib.h>

int main()

{

int i, j;

int a[2][3] = { 1,3,5,7,9,11 };

int b[2][3] = { {1,3,5},{7,9,11} };

int c[2][3] = { 1,3,5,7 };

int d[2][3] = { {1,3},{5,7} };

// int e[2][3]={{1,3},{5,7},{9,11}};

for (i = 0; i < 2; i++) {

printf("\n");

for (j = 0; j < 3; j++)

printf(" a[%d][%d]=%d", i, j, a[i][j]);

}

for (i = 0; i < 2; i++) {

printf("\n");

for (j = 0; j < 3; j++)

printf(" b[%d][%d]=%d", i, j, b[i][j]);

}

for (i = 0; i < 2; i++) {

printf("\n");

for (j = 0; j < 3; j++)

printf(" c[%d][%d]=%d", i, j, c[i][j]);

}

for (i = 0; i < 2; i++) {

printf("\n");

for (j = 0; j < 3; j++)

printf(" d[%d][%d]=%d", i, j, d[i][j]);

}

printf("\n");

system("pause");

return 0;

}

**Пример: /\* Умножение двух матриц \*/**



#include<stdio.h>

#include <windows.h>

#include<stdlib.h>

int main()

{

int A[3][4] = { {1,2,5,3},

{2,4,0,1},

{3,-1,2,4}

};

int B[4][2] = { {2,-1},

{3,2},

{3,4},

{2,-2}

};

int C[3][2];

int i, j, k;

for (i = 0; i < 3; i++)

for (j = 0; j < 2; j++) {

C[i][j] = 0;

for (k = 0; k < 4; k++)

C[i][j] += A[i][k] \* B[k][j];

}

for (i = 0; i < 3; i++) {

for (j = 0; j < 2; j++) {

printf("C[%d][%d]=%d\t", i, j, C[i][j]);

}

printf("\n");

}

system("pause");

return 0;

}

// массивы и указатели

int array[5]={1,2,3,4,5};

int \*iPtr;

iPtr=array; // iPtr=&array[0];

printf("\n\n\*iPtr=%d array[0]=%d\n",\*iPtr,array[0]);

// доступ к данным

printf("\narray[3]=%d \*(array+3)=%d iPtr[3]=%d \*(iPtr+3)=%d\n",array[3],\*(array+3),iPtr[3],\*(iPtr+3));

#include<stdio.h>

#include <windows.h>

#include<stdlib.h>

int main()

{

//определение указателей

int i = 100;

int \*iPtr;

iPtr = &i;

printf("iPtr=%p adr &i=%p\n", iPtr, &i);

printf("\*iPtr=%d i=%d\n", \*iPtr, i);

int \*iPtr1, \*iPtr2, \*a;

// унарные операции

int \*iPtr0, s = 5;

iPtr0 = &s;

printf("iPtr0=%p\n", iPtr0);

iPtr0++;

printf("iPtr0=%p\n", iPtr0);

iPtr0--;

printf("iPtr0=%p\n", iPtr0);

// бинарные операции

int i1, j1;

a = iPtr0;

iPtr1 = a + 4;

iPtr2 = a + 9;

i1 = iPtr1 - iPtr2;

j1 = iPtr2 - iPtr1;

printf("i1=%d j1=%d\n", i1, j1);

// операторы сравнения

int \*iPtr3, \*iPtr4, \*z, z0 = 10;

z = &z0;

iPtr3 = z + 9;

iPtr4 = z + 7;

if (iPtr3 > iPtr4) \*z = 4;

printf("z0=%d\n", \*z);

system("pause");

return 0;

}

**Строковые литералы, массивы и указатели**

Строковые литералы —последовательность символов, заключенная в двойные кавычки. В строковом литерале на один символ больше, чем используется при записи. Он всегда заканчивается нулевым символом **'\0’**.

**Пример:**

sizeof("Бор"); // значение 4

Строковый литерал можно присвоить переменной типа **char\***

Но, изменение литерала через такой указатель является ошибкой.

**Пример:**

#include<stdio.h>

#include <windows.h>

#include<stdlib.h>

int main()

{

char cStr[] = "Mars";

cStr[2] = 'p'; // ошибки нет

printf("cStr=%s\n", cStr);

char\* pStr = "Mars"; //

// pStr[2] = 'p';// ошибка при выполнении программы

printf("pStr=%s\n", pStr);

system("pause");

return 0;

}