Основы программирования

Указатели и динамические массивы

Описание и создание переменных

Простое описание числовых переменных:

```
int k; double x;
```

После создания глобальные переменные по умолчанию инициализируются нулем, а для остальных содержимое выделенной памяти не изменяется.

Явная инициализация при описании:

```
int k = 4; double x = 3.1415;
```

Раздельное описание и присвоение начального значения:

```
int k; double x;
```

```
k = 4; x = 3.1415; или cin >> k >> x;
```

Указатели

Указатель – это переменная (константа), значением которой является адрес области памяти, выделенной для переменной или значения **определенного типа**.

Общий формат описания переменной-указателя:

```
тип *имя_переменной;
```

Пример:

```
int *pa; double *px;
```

pa будет хранить **адрес** переменной/значения типа **int**

px – **адрес** переменной/значения типа **double**.

pa и **px** будет выделена память, необходимая для хранения адреса в памяти (4 или 8 байт), но реальные адреса по умолчанию не задаются.

Значения переменным-указателям нужно присваивать явно!

Операции с указателями

Взятие адреса – унарный префиксный **&** – извлечение адреса **переменной** (константы хранятся в памяти, но их адреса недоступны в программе):

```
int a = 2, b, *pa;
double y, x = 3.1415, *px;
pa = &a; px = &x;
```

Разыменование – унарная префиксная * – ссылка на объект, на который указывает указатель:

```
y = *px; b = (3 + *pa) * 10;
cin >> *pa; cout << *px;
*pa *= 5; *px = y * 0.01;
```

Арифметические операции

К указателям можно применять только целочисленное **сложение** или **вычитание**. При увеличении (уменьшении) указателя на 1 происходит переход не к следующему (предыдущему) байту, а к **следующему (предыдущему) элементу заданного типа**:

```
int *pa, a = 2; double x = 3.1415, *px;
pa = &a; px = &x;
```

`px++;` // **px** указывает на следующий за **x** в памяти элемент **double** (**px** увеличивается на 8)

`*(pa+5) = 7;` // 5-му после **a** в памяти элементу **int** (на расстоянии 5*4 байт от **a**) присваивается значение 7, значение **pa** не изменяется

Операторы для работы с памятью

Динамическое выделение памяти:

`указатель = new тип;` // выделяется память, необходимая для **`типа`**, адрес начального байта присваивается **`указателю`**.

Освобождение памяти:

`delete указатель;` // освобождается память, выделенная оператором **`new`** для **`указателя`**.

Примеры:

```
int *pa; double *px, r;
pa = new int; px = new double;
cin >> *pa; *px = 3.1415;
cout << *pa * *pa; *pa = (int)*px;
delete px; delete pa;
```

Статические и динамические массивы

Описание статического одномерного массива:

`int arr[50];` // имя статического массива **arr** – это **указатель-константа**, хранящий адрес начального байта массива

Динамическое выделение памяти для массива:

`указатель = new тип[длина];` – выделяется память, необходимая для **длина** элементов **типа**, адрес начального байта присваивается **указателю**. **длина** - это любое целочисленное выражение, константа или переменная.

Освобождение памяти, занятой массивом:

`delete [] указатель;` – освобождается память, выделенная оператором **new** для массива.

Статические и динамические массивы

Описание и выделение памяти:

```
int arr[50]; double *mas;
mas = new double [100];
```

Обращение к элементам (в любых массивах нумерация элементов начинается с 0):

`mas[0]` эквивалентно `*mas`

`arr[i]` эквивалентно `*(arr+i)`

`mas[j]` эквивалентно `*(mas+j)`

`arr+i` – адрес `i`-го элемента `arr`

`mas+j` – адрес `j`-го элемента `mas`

Освобождение динамического массива:

```
delete [] mas;
```

Пример динамического массива

```
int k, n; double *arr, *ptr;
cout << "Input array length: ";
cin >> n;
arr = new double[n];
srand(time(0));
for (k = 0; k < n; k++)
    arr[k] = (double)(rand()) / RAND_MAX;
for (ptr = arr, k = 0; k < n; k++, ptr++)
    cout << *ptr << endl;
delete [] arr;
```

Указатель на указатель

Переменная-указатель хранится в памяти, как и все переменные других типов, т.е. имеет определенный адрес. Для хранения адреса указателя нужно использовать переменную типа указатель на указатель (двойной указатель):

```
int x, *px, **ppx;
x = 13;
px = &x;      // px получает адрес x
ppx = &px;    // ppx получает адрес px
*ppx          // px или ее значение
**ppx, *px    // x или ее значение 13
```

Двумерный массив и указатели

Одномерный **статический** массив

```
int mas[50];
```

Здесь **mas** является константой-указателем на начальный (нулевой) элемент массива.

Двумерный **статический** массив

```
int arr[10][20];
```

Здесь **arr** имеет специальный тип:
int (*)[20] – это указатель на целочисленный массив длиной 20 (массив массивов).

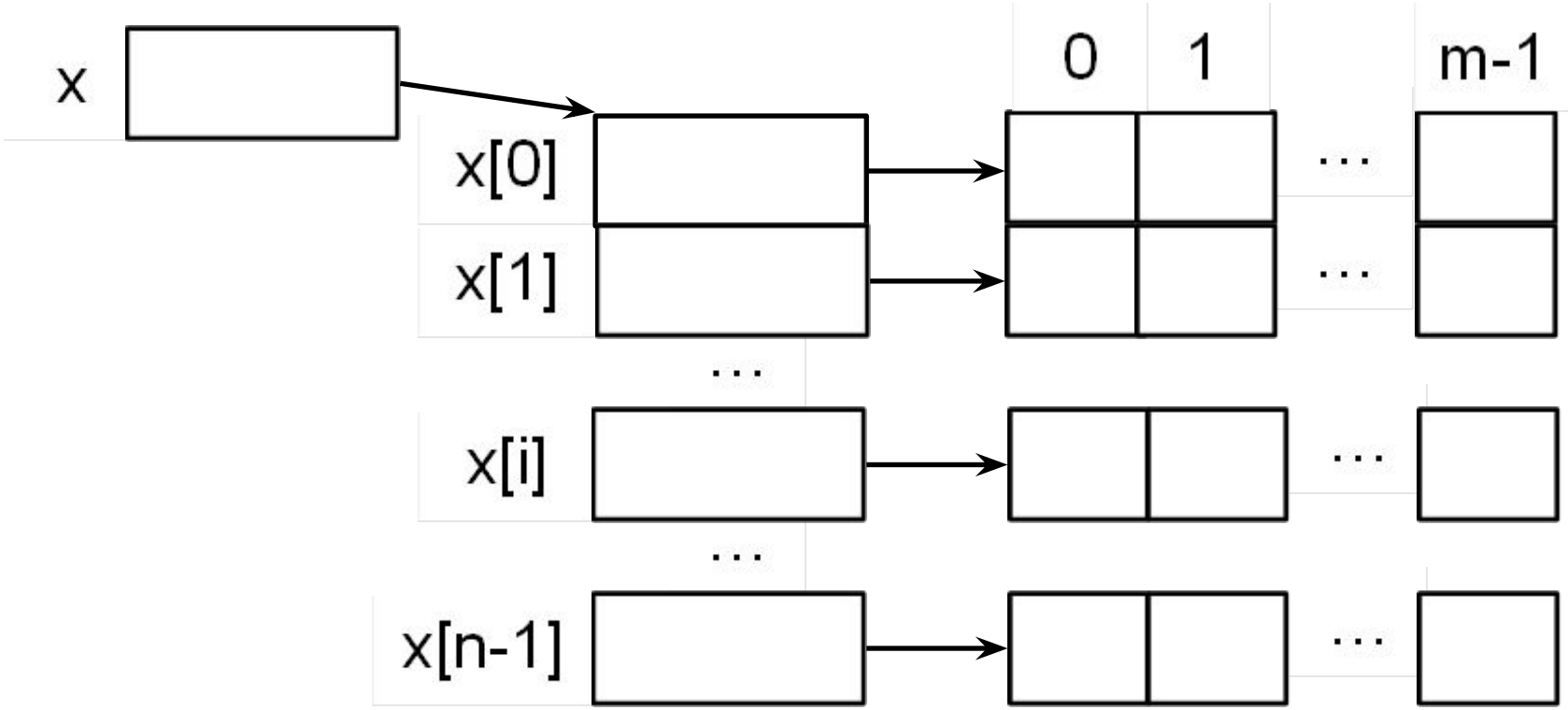
arr не является указателем на указатель **int ****.

10 строк по 20 элементов располагаются в памяти последовательно.

Двумерный массив и указатели

При создании двумерного **динамического** массива с **n** строками и **m** столбцами необходимо явно выделить память для:

- массива длины **n** указателей на строки
- **n** массивов длины **m** для элементов строк.



Двумерный динамический массив

```
int n, m, **x, i, j;
cin >> n >> m;

x = new int* [n];   // массив указателей

for (i = 0; i < n; i++)
   x[i] = new int [m];   // массивы целых

for (i = 0; i < n; i++)
   for (j = 0; j < m; j++)
      x[i][j] = rand();

for (i = 0; i < n; i++)
   delete [] x[i];

delete [] x;
```

Матрица в виде одномерного массива

```
int n, m, *x, i, j;
cin >> n >> m;

x = new int [n*m];          // массив целых

for (i = 0; i < n; i++)
    for (j = 0; j < m; j++)
        x[i*m+j] = rand();

delete [] x;
```

Области видимости переменных

```
int a = 100;
double fun(int d, int t)
{
   return a / d + t;
}
int _tmain(int argc, _TCHAR* argv[])
{
   float a = 3.1415; double *pm;
   int d = 123, t = 456, k = 0;
   pm = new double [2];
   for (int a = 0; a < 5; a++) k +=a;
   pm[0] = k + a;
   pm[1] = fun(d, t);
   cout << pm[0] << " " << pm[1];
   return 0;
}
```

Области памяти для переменных

4 основные области памяти, выделяемые программе:

1. **Область кода** – собственно программа
2. **Область данных** (data) – глобальные и **static**-переменные
3. **Стек** (stack) – локальные переменные всех блоков, параметры функций
4. **Куча** (heap) – область динамически выделяемой памяти