Язык программирования C++

Лекция 5

- Указатели
- Динамические массивы
- Область видимости переменных
- Формальные аргументы функций

Указатели

Указатель — это объект, который указывает на другой объект, т. е. содержит адрес этого другого объекта

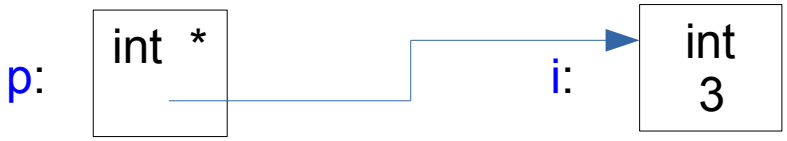
```
int*  p;
int  *q;
```

Допускаются обе формы записи, но вторая предпочтительней

Присвоение значения указателю:

```
int  i = 3;
int  *p = &i;
```

& - оператор взятия адреса



Внимание:

```
int  *p, i;
p = &i;
```

i – имеет тип int

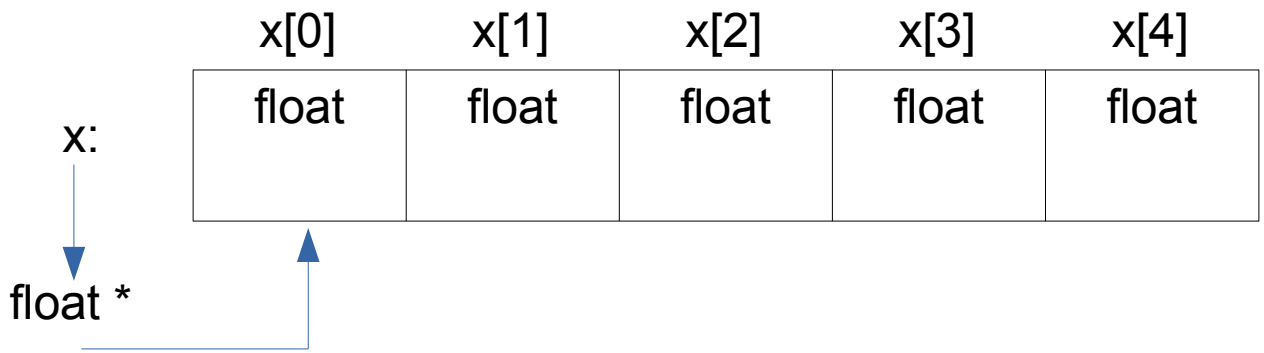
Разыменование указателей

```
#include <iostream>
using namespace std;

int main()  {
        int*  p;
        int  j = 4;
        p = &j;
        cout << *p << endl;

        *p = 5;
        cout << *p << " " << j << endl;
        if (p != 0) {
            cout << "Pointer p points at " << *p << endl;
        }
        return 0;
}
```

- *p – операция разыменования ("dereference") указателя для доступа к объекту
- *p может быть использован слева и справа от знака присвоения
- если p равен 0, то это нуль-пойнтер
- разыменовывание нуль-пойнтера приводит к аварийной остановке программы

Указатели и массивы

Рассмотрим массив: `float x[5];`



- в C/C++ x является указателем на первый элемент массива
- *x и x[0] – одно и то же
- x и &x[0] – одно и то же
- доступ к элементам массива может осуществляться любым способом
- но x остается идентификатором массива, и к нему не применяются правила работы с указателями

Арифметика указателей

Доступ к элементам массива может осуществляться с помощью указателей:

```
float  x[5];
float  *y = &x[0];
float  *z = x;
```

- y является указателем на x[0]
- z также является указателем на x[0]
- y+1 является указателем на x[1]
- поэтому *(y+1) и x[1] осуществляют доступ к одному и тому же объекту
- y[1] – это сокращенная запись *(y+1)
- с указателями можно совершать целочисленные операции сложения, вычитания и сравнения

Примеры (1)

Суммирование элементов массива без использования указателей в стиле языка Фортран:

```
float x[5];
double sum;
int i;
// … операторы заполнения элементов массива x
sum = 0.0;
for (i = 0;  i < 5;  i = i + 1) {
    sum = sum + x[i];
}
```

Суммирование элементов массива в стиле C++:

```
float x[5];
// … операторы заполнения элементов массива x
double sum = 0.0;
for (int i = 0;  i < 5;  i++) {
    sum += x[i];
}
```

- sum декларируется непосредственно перед использованием
- sum инициализируется в момент декларирования
- используется i++ для наращивания переменной цикла
- используется sum += для подсчета суммы элементов

Примеры (2)

Суммирование элементов массива с использованием указателей в стиле языка Фортран/С:

```
float x[5];
float *y;
double sum;
int i;
// … операторы заполнения элементов массива x
sum = 0.0;
y = x;
for (i = 0;  i < 5;  i = i + 1) {
    sum = sum + *y;
    y = y + 1;
}
```

Суммирование элементов массива с использованием указателей в стиле C++:

```
float x[5];
// … операторы заполнения элементов массива x
float *y = x;
double sum = 0.0;
for (int i = 0;  i < 5;  i++) {
    sum += *y++;
}
```

- декларирование переменных производится непосредственно перед их использованием
- используются операторы инкремента
- используется оператор сложения с инкрементом +=

Примеры стилей записи

Стиль “Фортран”

```
sum = sum + *y;
y = y + 1;
```

Используем оператор присвоения со сложением

```
sum += *y;
y = y + 1;
```

Используем оператор постфиксного инкремента

```
sum += *y;
y++;
```

Комбинируем постфиксный инкремент и разыменовывание

```
sum += *y++;
```

- понимание такого стиля записи требует времени
- особенно при разборе чужих программ
- эти стили записи не влияют на производительность программы

Примеры арифметики указателей

Обратный порядок элементов в массиве

```
float x[10];
// … операторы заполнения элементов массива x
float  *left = &x[0];
float  *right = &x[9];
while (left < right) {
    float temp = *left;
    *left++ = *right;
    *right-- = temp;
}
```

Зануление элементов массива

```
float x[10];

float  *p = &x[10];      //  !?
while (p != x) *--p = 0.0;
```

- в физике редко используется такая сложная для понимания запись
- обычно это стиль опытных профессиональных программистов на C/C++

Динамические массивы

В C++ можно динамически задавать массивы

```
float  *x = new float[n];
```

- new – оператор языка C++, который возвращает указатель на вновь созданный объект
- n – переменная, определяющая размерность массива
- в языке C та же операция проводилась с использованием системной функции malloc( )

```
float  *x = (float *)malloc( n*sizeof(float) );
```

В языке C++ для удаления динамически созданного массива используется оператор delete

```
delete  [ ]  x;
```

В языке C для использовалась системная функция free( )

```
free (x);
```

Область видимости (1)

```
void  func ()  {
        float temp = 1.1;
        int a;
        int b;
        std::cin >> a >> b;

        if (a < b)
        {       int temp = a;     //это локальная переменная temp
                std::cout << 2*temp << std::endl;
        }   // блок закончился, локальная переменная temp уничтожена
        else
        {       int temp = b;    //другая локальная переменная temp
                std::cout << 3*temp << std::endl;
        }
        std::cout << a*b + temp << std::endl;
}
```

- каждая пара фигурных скобок { } определяет новую область видимости
- любая переменная, определенная внутри области видимости, уничтожается по выходу из этой области

Область видимости (2)

```
for (int i = 0;  i < count;  i++)  {
        if (a[i] < 10) break;
}
std::cout << i << std::endl;
```

- переменная i определена внутри цикла for
- если нет другого определения i, то компилятор выдаст ошибку
- если i определена еще и вне цикла for, то такая ситуация может оказаться неопределенной: разные версии компиляторов C++ по-разному ее обрабатывают
- лучше избегать таких неоднозначных конструкций

Формальные аргументы функций

```
void  func (int i, float x, float *a)  {
        i = 100;
        x = 101.0;
        a[0] = 0.0;
}

int j = 1;
int k = 2;
float y[ ] = {3.0, 4.0, 5.0};
func (j, k, y);
```

- чему будет равна пременная j после вызова функции func( ) ?
- в языках C/C++ аргументы передаются по значению, поэтому j и k не изменятся
- i, x и a – формальные аргументы в области видимости функции func( )
- момент вызова func( ) компилятор генерирует код для инициализации аргументов:

```
int i = j;
float x = k;            // преобразование типа
float  *a = y;          // указатель на массив
```

- поэтому y[0] станет равным 0.0

Практическая задача (№ С5)

Написать программу линейного фита по методу наименьших квадратов (МНК):

- ввод с клавиатуры координат трех точек на плоскости (x_i, y_i), $i=1,2,3$
- методом наименьших квадратов через них надо провести прямую $y(x) = a*x + b$, где a и b – искомые коэффициенты
- МНК заключается в минимизации суммы квадратов отклонений заданных точек от искомой прямой $\Sigma(y(x_i) - y_i)^2$
- Оформить программу надо в виде головной функции *int main()* и вызываемой из нее функции *linefit()*, которая получает координаты точек и выдает обратно в *main()* вычисленные коэффициенты прямой