ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА ЭКОНОМИКИ

Учебная практика 2015-2016
Основы C++

День 1

Указатели и ссылки

Виды памяти

Переменные:

- Располагаются в оперативной памяти
- Характеризуются адресом в памяти и значением

Виды памяти

- Статическая
- Автоматическая (стек)
- Динамическая (куча)

Статическая память

Статическая память — это область памяти, *выделяемая при запуске программы* до вызова функции *main* из свободной оперативной памяти для размещения глобальных и статических объектов, а также объектов, определённых в пространствах имён.

Объект называют **глобальным**, если он определён вне функции, класса или пространства имён.

Объект, определённый с использованием ключевого слова *static*, называют **статическим**

Глобальные объекты, а также объекты, определённые в пространствах имён или статически в классах, размещаются в памяти (конструируются) до вызова функции *main*, а разрушаются после завершения работы этой функции

Пространства имён — это способ разделения программы на логические составляющие; механизм ограничения области видимости имён

Автоматическая память

Автоматическая память — это специальный регион памяти, *резервируемый при запуске программы* до вызова функции *main* из свободной оперативной памяти и *используемый в дальнейшем* для размещения локальных объектов: объектов, определяемых в теле функций и получаемых функциями через параметры в момент вызова. Автоматическую память часто называют **стеком**

Непосредственное управление автоматической памятью — выделение памяти под локальные объекты при их создании и освобождение занимаемой объектами памяти при их разрушении — осуществляется компилятором.

Период времени от момента создания объекта до момента его разрушения, т. е. период времени, в течение которого под объект выделена память, называют **временем жизни объекта**

Динамическая память

Доступную программе свободную оперативную память часто, в связи со спецификой размещения в ней объектов, называют **кучей** (heap).

В общем случае нет чёткого, наперёд заданного (как в стеке) порядка расположения объектов в ней: распределение блоков памяти происходит динамически — в большинстве реализаций под объект отводится первый подходящий по размеру свободный блок. Вследствие этого динамическую память часто определяют как память, выделяемую из кучи (непосредственно во время выполнения программы под размещение конкретных объектов).

В отличие от стека, которым управляет компилятор, управление динамической памятью осуществляется явным образом:

- выделение памяти производится оператором *new*,
- освобождение — оператором *delete*

Указатели

Указатели предназначены для хранения адресов памяти и обращения к содержимому адреса.

Указатели — это особый вид *объектов*, предназначенных для хранения информации о местоположении в памяти других объектов

Занимают в памяти 4 байта (адрес в 32-битной ОС)

Указатели типизированные

Указатели используют для работы с динамической памятью.

Обозначения (символы) * и &

При описании переменных:

Символ * перед именем переменной означает, что переменная является указателем;

Символ **&** перед именем переменной означает, что переменная является ссылкой;

В выражениях (не при описании переменных):

Символ * перед именем переменной означает операцию разыменования (разадресации) указателя, т.е. берется значение переменной из ячейки, на которую указывает указатель;

Символ **&** перед именем переменной означает, что берется адрес переменной;

Указатели объявление и инициализация

```
double a;
double *pa;
pa = &a;
int b;
int* pb = &b;
double c = *pa;
```

Объявление указателя *pa на переменную типа double

Взятие адреса переменной a

Можно сразу объявить указатель и инициировать

Разыменование указателя (получение значения переменной a, на которую направлен указатель)

Указатели. Пример 2

```
int *pn, n;
pn = &n;
*pn = 100;    //  значение переменной n= 100
(*pn)++;      //  значение переменной n= 101
pn++;         // значение указателя увеличивается на
              // 4 – количество байт на хранение int
pn = pn + 9; // значение указателя увеличивается на 4*9=36
```

Указатель – косвенное имя объекта

Таким образом, используя указатель pn и операцию разыменования, мы можем производить над объектом *n* в точности те же самые действия, что и с использованием имени *n*.

Следовательно, *pn* можно *трактовать* как ещё одно имя объекта *n*.

Подобные имена называют **косвенными**.

В итоге у нас есть *один* целочисленный объект и *два* его имени:

- прямое — *n*,
- косвенное — *pn*,

посредством которых можно обращаться к этому объекту — получать или изменять его состояние.

Косвенных имён может быть и больше.

Двойственность указателя

У объекта-указателя, равно как и у любого другого объекта, есть адрес — номер ячейки, начиная с которой объект-указатель располагается в памяти

В большинстве случаев сложности при работе с указателями возникают вследствие двойственности их природы: можно работать с адресом, являющимся значением указателя, а можно с объектом, расположенным по этому адресу

Указатели – пример 3

```
#include <iostream>
using namespace std;
void main() {int a = 100; int b = 200; int *pa = &a; int *pb = &b;
   cout<<"1 pa ="<<pa<<"  *pa="<<*pa<<"  a="<<a<<" b="<<b<<endl; //1
   cout<<"2 pb ="<<pb<<"  *pb="<<*pb<<"  a="<<a<<" b="<<b<<endl; //2
  pb=pa;
   cout<<"3 pb ="<<pb<<"  *pb="<<*pb<<"  a="<<a<<" b="<<b<<endl; //3
  pa = &b;
   cout<<"4 pa ="<<pa<<"  *pa="<<*pa<<"  a="<<a<<" b="<<b<<endl; //4
   cout<<"5 pb = "<<pb<<"  *pb="<<*pb<<"  a="<<a<<" b="<<b<<endl;//5
   cout<<"6 &pa= "<<&pa<<" &pb="<<&pb<<" &a="<<&a<<"&b="<<&b<<endl;
//6
  pa += 2;
   cout<<"7 pa = "<<pa<<" *pa="<<*pa<<" a="<<a<<" b="<<b<<endl; //7
   cout<<"8 "<<&pa<<"  &pb="<<&pb<<" &a="<<&a<<" &b="<<&b<<endl;//8
 return; }
```

Указатели - пример

```
int a = 100; int b = 200; int *pa = &a; int *pb = &b;
cout<<"1 pa = "<<pa<<" *pa="<<*pa<<"a="<<a<<“b="<<b<<endl;//1
cout<<"2 pb ="<<pb<<" *pb="<<*pb<<" a="<<a<<" b="<<b<<endl;//2
cout<<"2_2 &pa = "<<&pa<<"  &pb="<<&pb<<endl;     //2_2
```

переменная	pa (&a)	pb (&b)	a (*pa)	b (*pb)
Ее адрес	0043FA20	0043FA14	0043FA38	0043FA2C
Ее значение	0043FA38	0043FA2C	100	200

```
1 pa = 0043FA38  *pa=100  a=100 b=200
2 pb = 0043FA2C  *pb=200  a=100 b=200
2_2 &pa = 0043FA20  &pb=0043FA14
```

Все ячейки расположены в памяти последовательно, через С (12) байт

Указатели - пример

```
pb=pa;
   cout<<"3 pb = "<<pb<<"  *pb="<<*pb<<"  a="<<a<<" b="<<b<<endl;   //3
```

```
3 pb = 0043FA38  *pb=100  a=100 b=200
```

переменная	pa (&a)	pb (&a)	a (*pa) (*pb)	b
адрес	0043FA20	0043FA14	0043FA38	0043FA2C
значение	0043FA38	0043FA38	100	200

Указатель `pb` теперь хранит адрес ячейки `a`

Указатели - пример

```
pa = &b;
cout<<"4 pa = "<<pa<<" *pa="<<*pa<<" a="<<a<<" b="<<b<<endl;//4
cout<<"5 pb = "<<pb<<" *pb="<<*pb<<" a="<<a<<" b="<<b<<endl;//5
cout<<"6 &pa= "<<&pa<<" &pb="<<&pb<<" a="<<&a<<"&b="<<&b<<endl;
//6
```

переменная	pa (&b)	pb (&a)	a (*pb)	b (*pa)
адрес	0043FA20	0043FA14	0043FA38	0043FA2C
значение	0043FA2C	0043FA38	100	200

```
4 pa = 0043FA2C  *pa=200  a=100 b=200
5 pb = 0043FA38  *pb=100  a=100 b=200
6 &pa= 0043FA20 &pb=0043FA14 &a=0043FA38 &b=0043FA2C
```

Указатель pa теперь хранит адрес ячейки b

Адреса ячеек не меняются, меняются значения переменных

Указатели - пример

```
pa += 2;
cout<<"7 pa ="<<pa<<" *pa="<<*pa<<" a="<<a<<" b="<<b<<endl;//7
cout<<"8 "<<&pa<<" &pb="<<&pb<<" &a="<<&a<<" &b="<<&b<<endl;//8
```

переменная	pa (&b)	pb (&a)	a (*pb)	b (*pa)
адрес	0043FA20	0043FA14	0043FA38	0043FA2C
значение	0043FA34	0043FA38	100	200

```
7 pa = 0043FA34 *pa=-858993460 a=100 b=200
8 0043FA20  &pb=0043FA14 &a=0043FA38 &b=0043FA2C
```

Указатель pa теперь хранит адрес ячейки, сдвинутой относительно ячейки a на 2*4=8 байт. Что там хранится, неизвестно.

Адреса ячеек по-прежнему не меняются, меняются значения переменных

Выводы

Что можно делать с указателями:

- Арифметические операции сложения и вычитания. При этом используется размер типа, на который направлен указатель.

  В примере добавляется не 2, а 2*(размер типа)

- Указатель можно перенаправить на другую переменную
- Для направления указателя на переменную используется унарная операция взятия адреса
- Унарная операция разыменования (разадресации) дает значение переменной, на которую направлен указатель
- указатели можно сравнивать (операции сравнения)

Ссылки

Ссылки обязательно инициализируются при описании.

Ссылка – второе имя переменной.

Ссылка всегда разыменована.

Нет арифметики ссылок.

```
#include <iostream>
using namespace std;
void main()
{    int a = 2;
     int &sa = a;  // обязательно инициировать
     cout << "sa = " << sa << "  a = " << a << "  &a = " <<
          &a << "  &sa = " << &sa;
     cin >> a;
}
```

Ссылки навсегда привязаны к той переменной, на которую указывают. Поэтому их иногда называют псевдонимом или alias

Указатели и ссылки

Тип указателя и переменной, на которую он направлен, должны совпадать

Тип ссылки и переменной, на которую она ссылается, должны совпадать.

	Объявление	инициализация	разыменование	Перенаправить на другую переменную	Арифметические операции и операции сравнения
Указатель	`int * pa;`	`int k;` `pa = & k;`	`int a = *pa;`	можно	Да
Ссылка	`int &b = c`		Всегда разыменована	нельзя	Нет