Лабораторная работа №2
Указатели и классы

Цели и задачи лабораторной работы

- получение навыков программирования на языке C++;
- изучение основных принципов построения программного обеспечения по эмуляции работы аппаратного обеспечения.

Теоретический материал и задания для получения навыков и умений

Указатели

Указатель – это объект, содержащий адрес другого объекта и позволяющий косвенно манипулировать этим объектом.

Обычно указатели используются для работы с динамически созданными объектами, для построения связанных структур данных, таких, как связанные списки и иерархические деревья, и для передачи в функции больших объектов – массивов и объектов классов – в качестве параметров. Каждый указатель ассоциируется с некоторым типом данных, причем их внутреннее представление не зависит от внутреннего типа: и размер памяти, занимаемый объектом типа указатель, и диапазон значений у них одинаков . Разница состоит в том, как компилятор воспринимает адресуемый объект. Указатели на разные типы могут иметь одно и то же значение, но область памяти, где размещаются соответствующие типы, может быть различной.

Указатель обозначается звездочкой перед именем. В определении переменных списком звездочка должна стоять перед каждым указателем. Например:

```
long *lp, lp2;
```

В данном случае lp – указатель на объект типа long, а lp2 – объект типа long. Оператор разыменования (*) может отделяться пробелами от имени и даже непосредственно примыкать к ключевому слову типа. Поэтому приведенные определения синтаксически правильны и совершенно эквивалентны:

```
string *sp;
string* sp;
```

Указатели поддерживают ряд операций: присваивание, получение адреса указателя, получение значения по указателю, некоторые арифметические операции и операции сравнения.

1. Присваивание

   Указателю можно присвоить либо адрес объекта того же типа, либо значение другого указателя. Присвоение указателю адреса уже рассматривалось в прошлой теме. Для получения адреса объекта используется операция &:

```
int a = 10
int *pa = &a;   // указатель pa хранит адрес переменной a
```

   При этом указатель и переменная должны иметь один и тот же тип, в данном случае это тип int. Присвоение указателю другого указателя:

```
int a = 10;
int b = 2;

int *pa = &a;
int *pb = &b;

cout << "Variable a: address=" << pa
<< "\t value=" << *pa << endl;
cout << "Variable b: address=" << pb
<< "\t value=" << *pb << endl;

pa = pb;    // теперь указатель pa хранит адрес переменной b
cout << "Variable b: address=" << pa
<< "\t value=" << *pa << endl;
```

2. Разыменования указателя

   Операция разыменования указателя позволяет получить объект по адресу, который хранится в указателе.

```
int a = 10;
```

```
int *pa = &a;
int *pb = pa;

*pa = 25;

cout << "Value on pointer pa: " << *pa << endl;  // 25
cout << "Value on pointer pb: " << *pb << endl;  // 25
cout << "Value of variable a: " << a << endl;    // 25
```

Через выражение *pa мы можем получить значение по адресу, который хранится в указателе pa, а через выражение типа *pa = значение вложить по этому адресу новое значение. И так как в данном случае указатель pa указывает на переменную a, то при изменении значения по адресу, на который указывает указатель, также изменится и значение переменной a.

3. Адрес указателя

```
 int a = 10;
int *pa = &a;
std::cout << "address of pointer=" << &pa << std::endl;
// адрес указателя
std::cout << "address stored in pointer=" << pa << std::endl;
// адрес, который хранится в указателе - адрес переменной a
std::cout << "value on pointer=" << *pa << std::endl;
// значение по адресу в указателе - значение переменной a
```

Классы

Механизм классов в C++ позволяет пользователям определять собственные типы данных. По этой причине их часто называют пользовательскими типами. Класс может наделять дополнительной функциональностью уже существующий тип. Как правило, классы используются для абстракций, не отражаемых встроенными типами адекватно.

Определение класса состоит из двух частей: заголовка, включающего ключевое слово class, за которым следует имя класса, и тела, заключенного в фигурные скобки. Внутри тела объявляются данные-члены и функции-члены и указываются уровни доступа к ним. Таким образом,

тело класса определяет список его членов. Каждое определение вводит новый тип данных. Даже если два класса имеют одинаковые списки членов, они все равно считаются разными типами:

```
 class First {
   int memi;
   double memd;
};

class Second {
   int memi;
   double memd;
};

class First obj1;
Second obj2 = obj1;   // ошибка: obj1 и obj2 имеют разные типы
```

Тело класса определяет отдельную область видимости. Объявление членов внутри тела помещает их имена в область видимости класса. Наличие в двух разных классах членов с одинаковыми именами – не ошибка, эти имена относятся к разным объектам. После того как тип класса определен, на него можно ссылаться двумя способами:

- написать ключевое слово class, а после него – имя класса. В предыдущем примере объект obj1 класса First объявлен именно таким образом;
- указать только имя класса. Так объявлен объект obj2 класса Second из приведенного примера.

Оба способа сослаться на тип класса эквивалентны. Первый заимствован из языка C и остается корректным методом задания типа класса; второй способ введен в C++ для упрощения объявлений.

Данные-члены класса объявляются так же, как переменные. Например, у класса Screen могут быть следующие данные-члены:

```
 #include
class Screen {
   string             _screen;   // string( _height * _width )
   string::size_type  _cursor;   // текущее положение на экране
   short              _height;   // число строк
```

```
   short               _width;    // число колонок
};
```

Поскольку мы решили использовать строки для внутреннего представления объекта класса Screen, то член _screen имеет тип string. Член _cursor – это смещение в строке, он применяется для указания текущей позиции на экране. Для него использован переносимый тип string::size_type. Объявления данных-членов очень похожи на объявления переменных в области видимости блока или пространства имен. Однако их, за исключением статических членов, нельзя явно инициализировать в теле класса:

```
 class First {
   int    memi = 0;    // ошибка
   double memd = 0.0;  // ошибка
};
```

Пользователям, по-видимому, понадобится широкий набор операций над объектами типа Screen: возможность перемещать курсор, проверять и устанавливать области экрана и рассчитывать его реальные размеры во время выполнения, а также копировать один объект в другой. Все эти операции можно реализовать с помощью функций-членов. Функции-члены класса объявляются в его теле. Это объявление выглядит точно так же, как объявление функции в области видимости пространства имен. Например:

```
 class Screen {
public:
   void home();
   void move( int, int );
   char get();
   char get( int, int );
   void checkRange( int, int );
   // ...
};
```

Определение функции-члена также можно поместить внутрь тела класса:

```
 class Screen {
public:
   // определения функций home() и get()
```

```
   void home() { _cursor = 0; }
   char get() { return _screen[_cursor]; }
   // ...
};
```

Функция-член может быть перегруженной. Однако она способна перегружать лишь другую функцию-член своего класса. По отношению к функциям, объявленным в других классах или пространствах имен, функция-член находится в отдельной области видимости и, следовательно, не может перегружать их. Например, объявление get(int, int) перегружает лишь get() из того же класса Screen:

```
 class Screen {
public:
   // объявления перегруженных функций-членов get()
   char get() { return _screen[_cursor]; }
   char get( int, int );
   // ...
};
```

О классе говорят, что он определен, как только встретилась скобка, закрывающая его тело. После этого становятся известными все члены класса, а следовательно, и его размер. Можно объявить класс, не определяя его. Например:

```
 class Screen;   // объявление класса Screen
```

Это объявление вводит в программу имя Screen и указывает, что оно относится к типу класса.

Тип объявленного, но еще не определенного класса допустимо использовать весьма ограниченно. Нельзя определять объект типа класса, если сам класс еще не определен, поскольку размер класса в этом момент неизвестен и компилятор не знает, сколько памяти отвести под объект.

Однако указатель или ссылку на объект такого класса объявлять можно, так как они имеют фиксированный размер, не зависящий от типа. Но, поскольку размеры класса и его членов неизвестны, применять оператор разыменования (*) к такому указателю, а также использовать указатель или ссылку для обращения к члену не разрешается, пока класс не будет полностью определен.

Член некоторого класса можно объявить принадлежащим к типу какого-либо класса только тогда, когда компилятор уже видел определение этого класса. До этого объявляются лишь члены, являющиеся указателями или ссылками на такой тип. Ниже приведено определение StackScreen, один из членов которого служит указателем на Screen, который объявлен, но еще не определен:

```
 class Screen;   // объявление
class StackScreen {
   int topStack;
   // правильно: указатель на объект Screen
   Screen *stack;
   void (*handler)();
};
```

Поскольку класс не считается определенным, пока не закончилось его тело, то в нем не может быть данных-членов его собственного типа. Однако класс считается объявленным, как только распознан его заголовок, поэтому в нем допустимы члены, являющиеся ссылками или указателями на его тип. Например:

```
 class LinkScreen {
   Screen window;
   LinkScreen *next;
   LinkScreen *prev;
};
```

Задания для самостоятельного выполнения

Реализуйте заданный класс, в функции main должны быть представлены примеры работы всех методов класса. Аргументы методов передаются в качестве указателей.

1. Вектор. Хранит координаты конца вектора в n-мерном пространстве выходящего из начала координат. Методы: увеличение количества координат, уменьшение количества координат, сумма двух векторов, произведение двух векторов, разность двух векторов, умножение вектора на скаляр, длина вектора.

2. Строка. Хранит строку переменной длины. Методы: конкатенация двух строк, вырезание подстроки, сравнение двух строк на равенство, неравенство, меньше, больше.
3. Дата. Хранит день, месяц и год. Методы: разность в днях двух дат, дата отстоящая от заданной на заданное количество дней (месяцев), день недели.
4. Время. Хранит секунды минуты и часы. Методы: разность в секундах двух отрезков времени, время отстоящее от заданного на заданное количество секунд (минут).
5. Множество. Хранит элементы множества целого типа. Методы: принадлежность элемента множеству, добавить элемент, удалить элемент, является ли множество подмножеством данного, равны (не равны) ли два множества, объединение, пересечение, разность, симметрическая разность двух множеств.
6. Дробь. Хранит числитель и знаменатель обыкновенной дроби. Методы: сократить дробь, сумма, разность, произведение, деление двух дробей, равны ли две дроби, правильная ли дробь.
7. Матрицы. Хранит количество строк и столбцов, элементы целочисленной матрицы. Методы: сумма, разность, произведение двух матриц, умножение матрицы на скаляр, транспонирование матрицы, равны ли две матрицы, квадратная ли матрица, для квадратной матрицы – проверка симметричности относительно главной (побочной) диагонали.
8. Многочлены. Хранит список одночленов (коэффициент, степень, степень,...). Методы: принадлежит ли одночлен многочлену, сумма, разность, произведение, частное, остаток от деления двух многочленов, приведение подобных, равенство двух многочленов.

Содержание и оформление отчета

В отчет по лабораторной работе включается следующая информация:

1. Название лабораторной работы
2. Осноновные положения теоретического материала
3. Результаты выполения самостоятельных заданий в виде листингов программ.