Operator Overloading

Operator Overloading

- The compilers built in operator symbols work with classes defined by the programmer. The meaning of an operator may be overloaded (in OO languages), so that its behaviour may be polymorphic i.e. implemented differently for different classes.
- Overloading operators allow us to give all classes in a system a common interface, allowing us to perform similar operations on a range of different objects.

Assignment

- Assignment (=) is already overloaded. It has a default behaviour for all objects. It can be overridden for individual classes and not automatically for their descendant classes.
- The user defined behaviour of any other operator may be inherited by descendant classes.

C++ Syntax

```
Return_type operator symbol(parameter list)
```

- Operators which may be overloaded include arithmetic and relational operators.
- The return type for the boolean operators is a boolean value represented by 1 or 0.
- Arithmetic operators return type is an object of the same class as other objects in the expression.

- The parameter list usually consists of an object of the class, passed by reference rather that by value, to avoid making an unnecessary copy.
- The const prefix is frequently used to indicate that, although the object has been passed by reference it should not be modified by the method.

Overloading the '+' operator

```
class studentgrade
{private:
        int mathsgrade;
        int englishgrade;
public:
        studentgrade();
        int getmathsgrade();
        int getenglishgrade();
        void setmathsgrade(int grade_in);
        void setenglishgrade(int grade_in);
};
```


Adding student grades :

- Grade 3 = Grade 1 + Grade 2
- If we want to add 2 studentgrade objects together we must add the following to the above class:

```
  studentgrade operator + (const
  studentgrade& grade_in);
void main()
{
studentgrade grade_total, student1, student2;
grade_total = student1 + student2;
}
```


The operator definition

```
studentgrade studentgrade::operator + (const
  studentgrade& grade_in)
{
      studentgrade result;
      result.mathsgrade =
        mathsgrade + grade_in.mathsgrade;

      result.englishgrade =
      englishgrade + grade_in.englishgrade;

 return result;
}
```

- student1 calls the operator method
- student2 is its argument
- grade_total is its return value.

- In main we can reference the components of the object grade_total as follows:
- grade_total.getmathsgrade() …

Stream Classes

- A stream is a general name given to a flow of data e.g. cin, cout. In C++ a stream is represented by an object of a particular class.
- There are no formatting characters in streams, since each object knows how to display itself (due to operator overloading).
- Existing operators and functions (e.g. insertion << and extraction >>) may be overloaded to work with classes that you create.

Istream & ostream

- The istream class performs specific input activities or extractions
- The ostream class handles output or insertion activities
- To overload the ostream and istream operators we use the following syntax:

```
ostream& operator << (ostream&, class_type&);
istream& operator >> (istream&, class_type&);
```

where class_type is a parameter of some user defined class, passed by reference. The method returns a reference to an ostream / istream object, and also takes one as parameter.


```
class person{
private:
  char name[20];
  int age;
public:
  char* getname();
  int getage();
  void setname(char * name_in);
  void setage(int age_in);
};
```

The Methods

```
char* person::getname()
{ return name; }

int person::getage()
{ return age; }

void person::setname(char* name_in)
{ strncpy(name, name_in, 19);
   name[19] = '\0';}

void person:: setage(int age_in)
{ age = age_in; }
```


Overload >>

```
istream&operator >> (istream& in, person& person)
{  char temp[20];
   int age;
   cout << "Enter Age";
   in >> age;
   person.setname(name);
   person.setage(age);
   return in;
}
```


Overloading <<

```
ostream& operator <<
        (ostream& out, person& person)
{ return out << "Name"<<
  person.getname() <<endl
  <<"Age" << person.getage() <<endl;
}
```


Main

In main a person is input / output using cin and cout

```
void main()
{     person p;
      cin >> p;
      cout << p;
}
```

- Most programs need to save data to disk files and read it back. Working with disk files require the following:
  - ifstream for input
  - fstream for input and output
  - ofstream for output
- Objects of these classes can be associated with disk files, and we can use their member functions to read and write to files. These classes are declared in **fstream.h** which includes the iostream.h file

- **Reading data from a file** uses the ifstream object, initialised to the name of the file. The file is automatically opened when the object is created. We can read from it using the operator <<

ifstream infilename("fdata.txt");

infilename >> str1 >> str2;

cout <<str1<< str2;

- Writing data to a file uses the ostream object, initialised to the name of the file. The file is automatically opened when the object is created.
- We can write to it using the operator >>

ofstream outfilename("fdata.txt");

// fdata.txt is the file to be opened for reading/writing

- in / outfilename are ifstream/ostream objects e.g. an object of type file_of _borrower

- This will open the file on disk/ if it doesn't exist it creates the file.

Outfilename <<str1 << ' ' << str2

  - Outputs "str1 str2" to the file. When the program terminates the outfilename object goes out of scope.

Opening modes:

- ios::in opens for input (default for ifstream)
- ios::out opens for output (default for ofstream)
- ios.ate open and seek end of file
- ios::app append all output
- ios::trunc destroys contents of existing file
- ios::nocreate open fails if file doesn't exist
- ios::noreplace open fails if thefiel exists
- e.g. *ofstream outfilename("fdata.txt" , ios::app | ios::nocreate);*

Closing a file

– *filename.close();* closes the file filename

Detecting EOF

– *while (!infilename.eof())* // while the end of file has not been encountered

Detecting Errors

– *while (infilename.good())* // while an error has not been encountered

I/O of single characters

– put() and get() may be used for I/O for single characters

- *outfilename(put(A[i]);*
- *infilename(get(A[i]);*

Searching a file

- Read from the file
- Use strcmp to compare the file contents to the item you are searching for
  – E.g. The following code reads an object called person. The class that this object belongs to has a method called getname. This method returns a string which is compared to the string (str) which is been searched for.

```
person.read((char*)&person, sizeof person);
if ((strcmp person.getname(), str) == 0)
…
```

- **Writing to a file**

  cout << "\n Enter Person Data";
  pers.getdata();
  file.write((char) &person ,sizeof(person))*

- **Reading from the start of a file**

  file.seekg(o)
  file.read((char) &person, sizeof(person))*


```
// saves person object to disk
#include <fstream.h>        //for file streams

class person               // class of persons
  { private:
      char name[40];         // person's name
      int age;                 // person's age
   public:
     void getData()  // get person's data
     void showData()  // display person's data
   };
```



```
void Person ::getData()  // get person's data
   {
   cout << "Enter name: "; cin >> name;
   cout << "Enter age: "; cin >> age;
   }
void Person::showData()  // display person's
  data
   {
   cout << "\n   Name: " << name;
   cout << "\n   Age: " << age;
   }
```



```
void writing()
   {  person pers;     // create a person
      pers.getData();   // get data for person

                        // create ofstream object
     ofstream outfile("PERSON.DAT",
                              ios::binary);
     outfile.write((char*)&pers,sizeof(pers));
     // write to it
  }
```

```
void reading()
   {  person pers;
                // create person variable

      ifstream infile("PERSON.DAT",
                      ios::binary);

                      // create stream
      infile.read((char*)&pers, sizeof(pers) );
                            // read stream
      pers.showData();
                      // display person
}
```