Object-oriented Programming

Filing

CREATED BY: SYED ZAIN UL HASSAN

Defining Aliases

- The keyword **typedef** can be used to declare synonyms (aliases) for previously defined data types

- Creating a name using *typedef* does not create a new data type; typedef creates only an alternate name for the existing data type

Example

```
int main()
{
    typedef int i;
    i var1 = 5;
    cout << var1;       // outputs 5

    typedef float f;
    f var2 = 2.8;
    cout << var2;       // outputs 2.8
}
```

Stream I/O

- C++ is a type-safe language
- I/O in C++ occurs in **streams**
- Streams are simply sequence of bytes
- In input operations, data is transferred from input device (keyboard) to main memory
- In output operations, data is transferred from main memory to output device (display screen)

Stream I/O

- C++ provides both low-level (unformatted) and high-level (formatted) I/O capabilities

- Unformatted I/O is efficient for high-volume data processing

- C++ includes the **standard stream libraries** for I/O operations

Stream I/O

- The C++ **iostream** library provides a lot of I/O capabilities
- The <iostream> header file defines the *cin*, *cout*, *cerr* and *clog* objects
- The <iomanip> header declares services useful for performing formatted I/O with so-called **parameterized stream manipulators**

Stream Operators

- The left-shift operator (**<<**) is overloaded to serve as stream insertion operator

- The right-shift operator (**>>**) is overloaded to serve as stream extraction operator

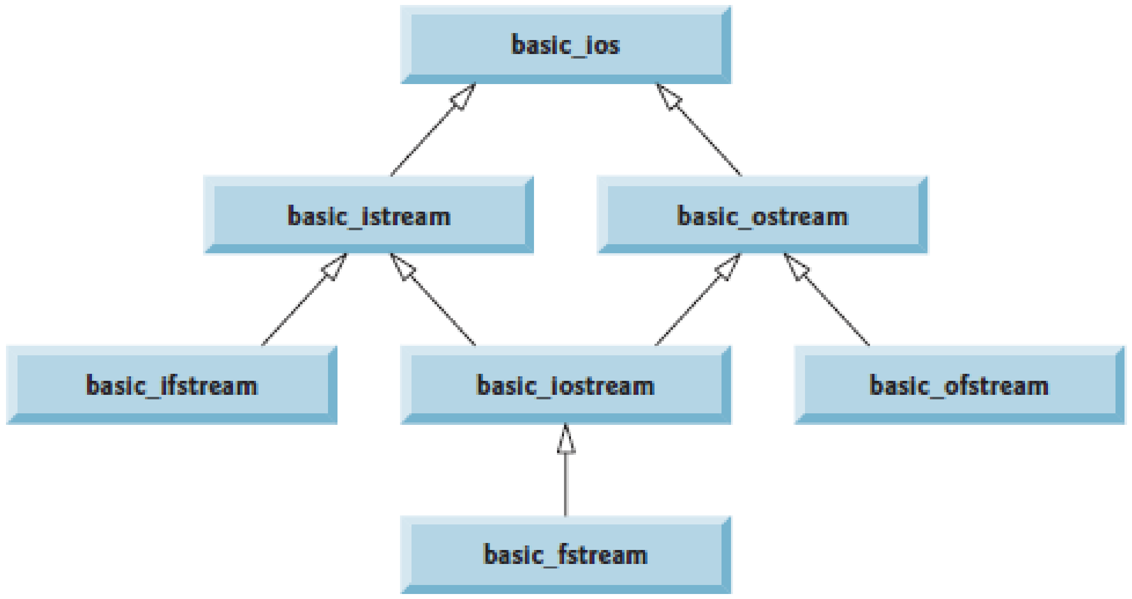
- These operators are used with the standard stream objects cin, cout, cerr and clog

Standard Stream Objects

- Predefined object **cin** is an istream instance
- The object **cout** is an ostream instance

- The predefined object **cerr** is an ostream instance and is said to be “connected to” the standard error device, normally the screen

- The predefined object **clog** is an instance of the ostream class and is said to be “connected to” the standard error device. Outputs to clog are **buffered**

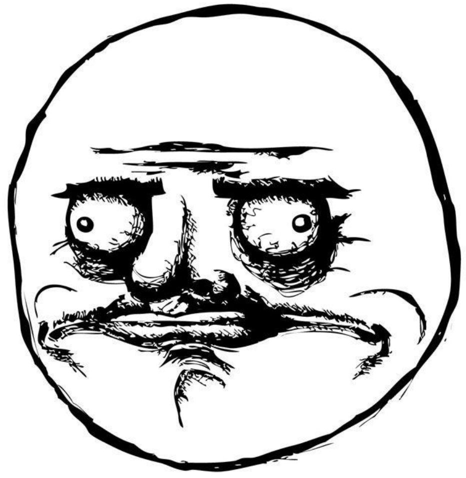


Stream-I/O template hierarchy portion showing the main file-processing templates.

Output Stream

- C++ determines data types automatically

- But this feature can sometimes *“gets in the way”*



Example

```
int main()
{
    char * word = “Hello”;
    cout << “Address is ” << word;
}
```

// prints Hello as output instead of address

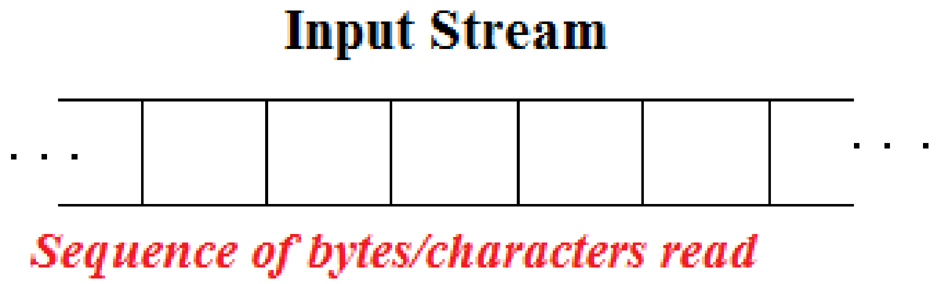
Solution

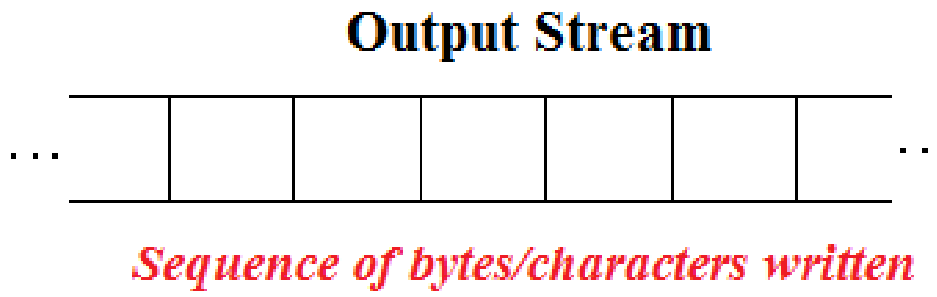
- Cast the char * to a void *
-

```
int main()
{
    char * word = “Hello”;
   cout << “Address is ”;
   cout << static_cast< void *>(word);
} // prints address as output
```

Stream

Stream (for us…)





istream/ostream Member Functions

- Both the ***istream*** and ***ostream*** classes provide member functions for input/output of data

- These member functions can be called using the ***iostream*** objects such as ***cin*** & ***cout***

Reading a character from file

get()

The **get()** member function of ifstream class reads a single character from the file input stream (including white-space and other characters like EOF) and returns it

Reading a character from file

get()

Example:

```
ifstream in(“xyz.txt”);
char c = in.get();
cout << c;
```

Reading a character from file

get(char)

The **get(char)** member function of *ifstream* class reads a single character from the file input stream (including white-space and other characters like EOF) in the character variable specified in the argument

Reading a character from file

get(char)

Example:

```
ifstream in("xyz.txt");
char c;
in.get(c);
cout << c;
```

Reading a line from file

getline(ifstream, string)

The function **getline(ifstream, string)** reads a single line from the file input stream specified as first argument and saves it in the string specified as the second argument

Reading a line from file

getline(ifstream, string)

Example:

```
ifstream in(“xyz.txt”);
string line;
getline(in, line);
cout << line;
```

Reading bulk text from a file

read(char*, int)

The function **read(char*, int)** reads n characters (including whitespaces and eof) into the buffer specified as first argument. The value of n is specified as second argument

Reading bulk text from a file

read(char*, int)

Example:

ifstream in(“xyz.txt”);

char text;*

int n = 20;

in.read(text, n);

Writing a character to file

put(char)

The **put(char)** member function of ofstream class writes a single character, taken as argument, to the file specified by file output stream object

Writing a character to file

put(char)

Example:

```
ofstream out("xyz.txt");
out.put('A');
char c = 'B';
out.put(c);
```

Writing bulk text to a file

write(char*, int)

The function **write(char*, int)** writes n characters from the char* buffer specified as first argument to the file. The value of n is specified as second argument

Writing bulk text to a file

write(char*, int)

Example:

```
ofstream out(“xyz.txt”);
out.write(“This is some text”, 8);
char* c = “This is some other text”;
out.write(c, 10);
```

istream Member Function

eof()

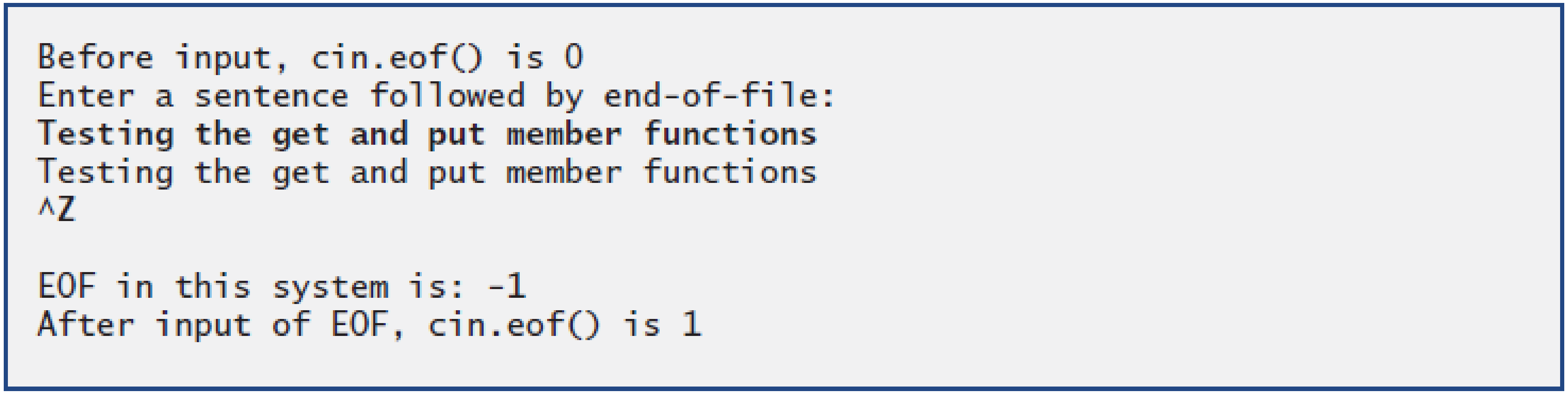
- It returns **1** (TRUE) when there are no more data to be read from an input stream, and **0** (FALSE) otherwise

```
int main()
{
int character;
cout << “Before input, cin.eof() is ”<< cin.eof();
cout << “Enter input followed by eof” <<  endl;

while((character = cin.get())  != EOF)
     cout.put( character );

cout << “EOF in this system is: ” << character;
cout << “After input, cin.eof() is: ” << cin.eof();
}
```

Output



```
Before input, cin.eof() is 0
Enter a sentence followed by end-of-file:
Testing the get and put member functions
Testing the get and put member functions
^Z

EOF in this system is: -1
After input of EOF, cin.eof() is 1
```

istream Member Function

ignore()

- The **ignore()** function of istream reads and discards a designated number of characters (the default is one) or terminates upon encountering a designated delimiter

istream Member Function

putback()

- The **putback()** function places the previous character obtained by a get from an input stream back into that stream

istream Member Function

peek()

- The **peek()** function returns the next character from an input stream but does not remove the character from the stream

Notes

- The default **end-of-file (EOF)** character sequence in Windows is ***Ctrl+Z***

- The default delimiter in most systems is **‘\n’** or newline character

Writing objects to file

```
class Employee
{
	char* name;
	int age;

	public:
	Employee() { }

	Employee (char* n, int a)
	{ name = n; age = a; }

	void display()
	{ cout <<  name << “ & “ << age << endl; }
}
```

Example (cont’d)

```
int main( )
{
    Employee  e1(“Ahsan”, 12400);
    Employee e2(“Ali”, 13500);

    ofstream os(“myfile.txt”, ios::app);
    os.write((char*)&e1, sizeof(e1));
    os.write((char*) & e2, sizeof(e2));
    os.close( );
}
```

Reading objects from a file

- We can read objects from a file using the ifstream member function read(char*, int)

- We can read an object and save it in an object declared beforehand

Example

```
int main( )
{
    Employee e;

    ifstream is(“myfile.txt”);
    is.read((char*)&e, sizeof(e));
    // use the object as you like
    is.close( );
}
```

Important Point

- To write/read objects to a file correctly, make sure your class has char array instead of string to store sequence of characters