Object Oriented Programming with C++

1. Introduction

Second Semester, 2021-22
Computer Engineering Department
Dharmsinh Desai University

What is C++

- Object Oriented Programming Language, developed by Bjarne Stroustrup in early 1980s
- Standardized in 1997, ANSI/ISO standard committee
- Most C programs are valid C++ programs
- Addition of OOP is the major change from C language

Applications of C++

- Versatile language
- System programming – Parts of Linux and Windows
- Application programming – Paint, Notepad etc.

- Now a days, very popular for competitive coding due to its good computing performance compared to java and python and avaibility of very good library compared to C language.

Sample C++ program

```
#include<iostream>
using namespace std;
/* Comments same as C Language */
int main() // Like C language, execution starts from main
{
    // cout is output stream object declared in iostream
    // It represents standard output
    // << is called insertion or put to operator. Declared in iostream
    // Sends contents of variable/constant on right to object on left
    cout << "Hello World!\n";
    return 0;
}
```

Explanation

- iostream file inclusion
  - No .h extension – old style don’t use it
  - For C library files prepend library name with ‘c’ and drop .h extension
  - e.g. #include<cstdio> instead of #include<stdio.h>

- using and namespace keywords
  - Will be covered in later lectures in more details

Another C++ program

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

int main()
{
    int num1, num2;

    cout << "Enter two numbers: ";
    cin >> num1 >> num2;

    int sum = num1 + num2; // Variable can be declared anywhere

    cout << "Sum is: " << sum << endl; // No need of format specifier
    return 0;
}
```

C Vs C++ sample program

```c
#include<stdio.h>

int main()
{
    int num1, num2, result;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    result = num1 + num2;

    printf("Result is %d\n", result);

    return 0;
}
```

```c
#include<cstdio>

int main()
{
    int num1, num2, result;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    result = num1 + num2;

    printf("Result is %d\n", result);

    return 0;
}
```

C Vs C++ sample program

```c
#include<cstdio>

int main()
{
    int num1, num2, result;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    result = num1 + num2;

    printf("Result is %d\n", result);

    return 0;
}
```

```c
#include<cstdio>
#include<iostream>

using namespace std;

int main()
{
    int num1, num2;

    cout << "Enter two numbers: ";
    cin >> num1;
    scanf("%d", &num2);

    int result = num1 + num2;

    printf("Result is %d\n", result);

    return 0;
}
```

C Vs C++ sample program

```c
#include<cstdio>
#include<iostream>

using namespace std;

int main()
{
    int num1, num2;

    cout << "Enter two numbers: ";
    cin >> num1;
    scanf("%d", &num2);

    int result = num1 + num2;

    printf("Result is %d\n", result);

    return 0;
}
```

```c
#include<iostream>

using namespace std;

int main()
{
    int num1, num2;

    cout << "Enter two numbers: ";
    cin >> num1 >> num2;

    int result = num1 + num2;

    cout << "Result is " << result << "\n";

    return 0;
}
```

Observations

- *cin* is input stream object declared in iostream and represents standard input
- >> is *Extraction* or *Get From* operator declared in iostream
- Cascading of I/O operators

- Operator overloading – bitwise shift Vs insertion/Extraction
  - Another example of operator overloading in C++
    - Use of + operator on string and numbers
  - Will be more clear later when we overload operators ourselves

- In C language bitwise AND (&) and address of (&) operators are same. But that is syntax of the language and not operator overloading.

- endl is manipulator defined in iostream

Compiling and running c++ program

- g++ prog.cpp
- ./a.out

Structure of C++ program

- So far its almost same as C program
- Will change once we introduce class concepts

Interesting reads...

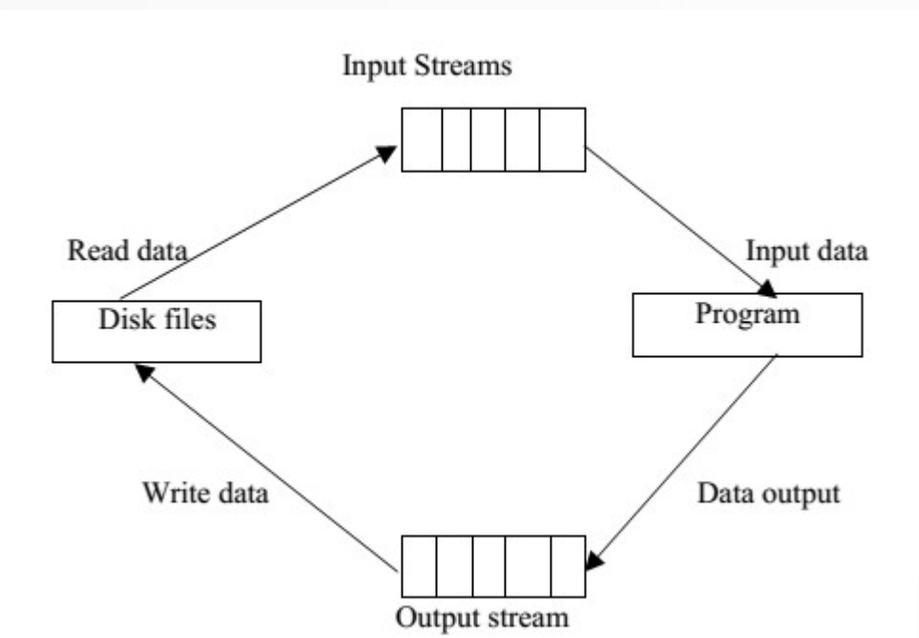
- Which is faster *cin* and *cout* Vs *scanf* and *printf*
  - https://stackoverflow.com/questions/1042110/using-scanf-in-c-programs-is-faster-than-using-cin
- “\n” vs endl
  - https://stackoverflow.com/questions/213907/c-stdendl-vs-n

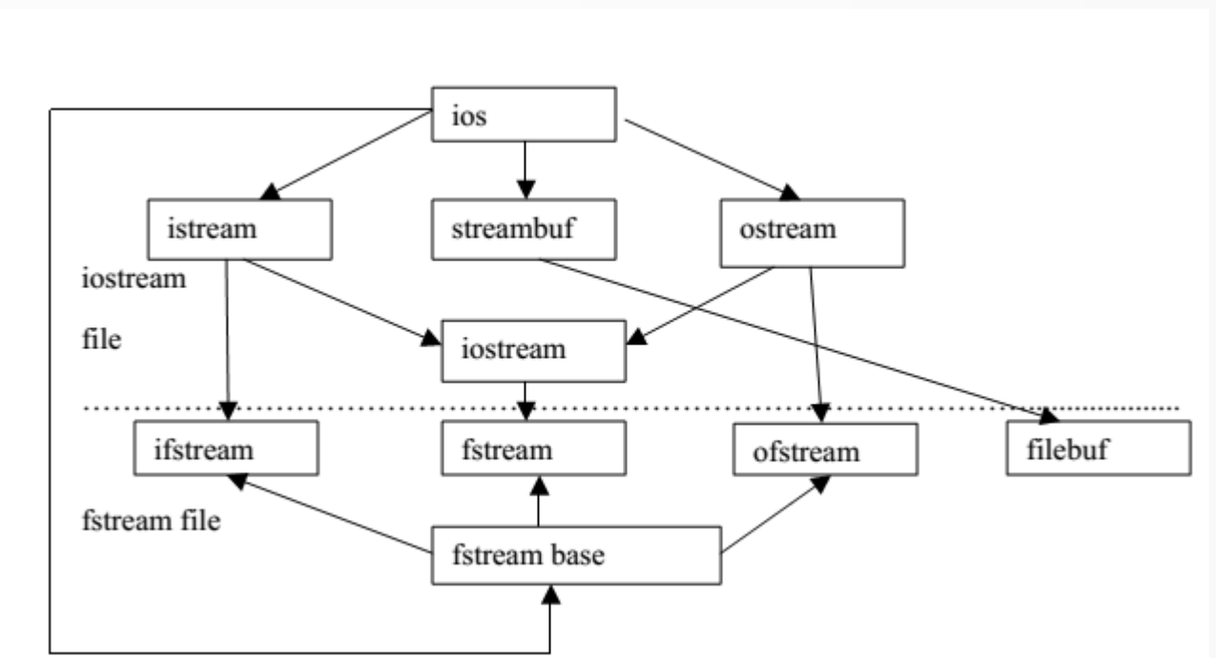
Object Oriented Programming with C++

10. C++ Managing Console I/O operations

Stream and Stream classes

- Stream
  - Sequence of bytes. It acts as source(input stream) or destination(output stream)
    - C++ contains several predefined streams that automatically opens when program begins





Unformatted I/O

- Overloaded operator << and >>
  - Operator >> reads data character by character.
    - Reading terminates :
      - At whitespace OR
      - At character that dose not match the destination type

```
#include<iostream>
int main()
{
    int data;
    std::cin>>data;
    return 0;
}
```

123Hello

- Here, data will contain 123 and Hello will be in buffer so it will be passed to next read

Unformatted I/O

- put() and get() functions
  - get() is istream function to read a single character.
  - put() is ostream function to output a single character.
  - char c;
  - cin.get(c); //c = cin.get(); alternative syntax
  - cout.put(c); //outputs the value of variable c
  - cout.put('c'); //outputs the character c

    cout.put(99); //outputs the character c (why???)

Unformatted I/O

- getline() and write() functions
- The getline() function reads a whole line of text that ends with a newline character.
- cin.getline(char_arr, int_size)
- The reading terminates as soon as either
  - The newline character ‘\n’ **OR**
  - size -1 characters are read
- char name[20];
- cin.getline(name,20);
- The write() function displays an entire line.
- cout.write(char_arr, int_size)
- It does not stop displaying the characters automatically when the null character is encountered. If the size is greater than the length of char_arr then it displays beyond the bounds of char_arr.
- It is possible to concatenate two strings using the write() function.
- cout.write(string_1, m).write(string_2, n); //(why???)

Formatted I/O

1) ios class functions and flags
2) Manipulators
3) User-defined output functions

Formatted I/O:ios class functions and flags

Function	Purpose
width ( )	To specify field size for displaying an output value
precision ( )	To specify the number of digits to be displayed after the decimal sign
fill ( )	To specify a character that fills the unused portion of an output data filed
setf ( )	To specify format flags such as left-justify, right-justify etc.
unsetf ( )	To clear/ reset defined flags

Formatted I/O:ios class functions and flags

The width() function can be used in two ways as shown below:
int width() : Returns the current width setting.
int width(int) : Sets the specified width and returns the previous width setting.

The precision() function can be used in two ways as shown below:
int precision() : Returns the current precision setting.
int precision(int) : Sets the specified precision and returns the previous precision setting.

The fill() function can be used in two ways as shown below:
char fill() : Returns the current fill character.
char fill(char) : Sets the fill character and returns the previous fill character.

The setf() function can be used in two ways as shown below:
long setf(long flag, long bit_field) : The bits specified by the variable bit_field are removed from data member x_flags in ios and set according the value specified by flag.
long setf(long) : Sets the flags in x_flags based on the given value.

Except width(), all others retain the setting in effect until it is reset.

Formatted I/O:ios class functions and flags

Flags and bit fields for setf() functions:

Format	Flag	Bit Field
Left justification	ios::left	ios::adjustfield
Right justification	ios::right	ios::adjustfield
Padding after sign and base	ios::internal	ios::adjustfield
Scientific notation	ios::scientific	ios::floatfield
Fixed point notation	ios::fixed	ios::floatfield
Decimal base	ios::dec	ios::basefield
Octal base	ios::oct	ios::basefield
Hexadecimal base	ios::hex	ios::basefield

Formatted I/O:ios class functions and flags

Flags without bit fields for setf() functions:

Flag	Purpose
Ios::showbase	Uses base indicator on output.
Ios::showpos	Displays + preceding positive number.
Ios::showpoint	Shows trailing decimal point and zeros.
Ios::uppercase	Uses capital case for output.
Ios::skipws	Skips white space on input.
Ios::unitbuf	Flushes all streams after insertion.
Ios::stdio	Adjusts the stream with C standard input and output.
ios::boolapha	Converts boolean values to text.

Formatted I/O:ios class functions and flags

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

// The width() function defines width
// of the next value to be displayed
// in the output at the console.
void IOS_width()
{


        char c = 'A';
        // Adjusting width will be 5.
        cout.width(5);
        cout << c <<"\n";
        int temp = 10;
        // Width of the next value to be
        // displayed in the output will
        // not be adjusted to 5 columns.
        cout<<temp;
        cout << "\n-------------------------\n";
}

void IOS_precision()
{
        cout.setf(ios::fixed, ios::floatfield);
        cout.precision(2);
        cout<<3.1422;
        cout<<"\t"<<3.145<<"\t"<<3.146;
        cout << "\n-------------------------\n";
}

// The fill() function fills the unused
// white spaces in a value (to be printed
// at the console), with a character of choice.
void IOS_fill()
{

        char ch = 'a';
        // Calling the fill function to fill
        // the white spaces in a value with a
        // character our of choice.
        cout.fill('*');
        cout.width(10);
        cout<<ch <<"\n";

        int i = 1;
        // Once you call the fill() function,
        // you don't have to call it again to
        // fill the white space in a value with
        // the same character.
        cout.width(5);
        cout<<i;
        cout << "\n-------------------------\n";
}
void IOS_setf()
{
        cout << "\n-------------------------\n";
        cout << "Implementing ios::setf\n\n";
        int val1=100,val2=200;
        cout.setf(ios::showpos);

void IOS_unsetf()
{
        cout << "\n-------------------------\n";
        cout << "Implementing ios::unsetf\n\n";
        cout.setf(ios::showpos|ios::showpoint);
        // Clear the showflag flag without
        // affecting the showpoint flag
        cout.unsetf(ios::showpos);
        cout<<200.0;
        cout << "\n-------------------------\n";
}

// Driver Method
int main()
{
        IOS_width();
        IOS_precision();
        IOS_fill();
        IOS_setf();
        IOS_unsetf();
        return 0;
}
```

C++ Operators - Manipulators

- Manipulators
  - If a pointer to function (which returns **ostream** reference and takes **ostream** reference as first argument) is given as the second argument to **<<** , the function pointed to is called. For example, **cout << pf** means **pf(cout)** . Such a function is called a manipulator.
  - **>>** needs - function which returns **istream** reference and takes **istream** reference as first argument – to work as manipulator for input streams.
  - manipulator is a function that can be invoked by **<<** or **>>**
  - Manipulators are used to format data
    - Can be used to format input and output streams. But frequently used to format output streams.
    - **Iomanip** contains declaration for many such manipulators
      - These manipulators internally call member functions of **ios** class

C++ Operators - Manipulators

- Manipulators (Only few important ones) **Explore more here**

Manipulator	Meaning
endl	Insert newline and flush stream
setw(int w)	Set field width to w. It only applies to next value to be inserted, then it is reset to default (0)
setfill(int c)	Set the fill character to c. Default is space.
left	Append fill characters at the end
right	Insert fill characters at the beginning
setprecision(int d)	Set the floating point precision to d.
fixed	Floating-point values are written using fixed-point notation: the value is represented with exactly as many digits in the decimal part as specified by the precision field

C++ Operators – Manipulators

```
#include<iostream>
#include<iomanip>
int main()
{
	std::cout << std::setw(10) << "Hello" << std::endl;
	std::cout << "Hello" << std::endl;
	std::cout << std::setw(2) << "Hello" << std::endl << std::endl;

	std::cout << std::left << std::setfill('*');
	std::cout << std::setw(9) << "Hello" << std::endl;
	std::cout << std::setw(7) << "Hello" << std::endl << std::endl;

	std::cout << std::setprecision(2) << 111.11111 << std::endl;
	std::cout << 1.11111 << std::endl;
	std::cout << std::setprecision(10) << 111.11111 << std::endl;
	std::cout << 1.11111 << std::endl << std::endl;

	std::cout << std::fixed;
	std::cout << std::setprecision(2) << 111.11111 << std::endl;
	std::cout << 1.11111 << std::endl;
	std::cout << std::setprecision(10) << 111.11111 << std::endl;
	std::cout << 1.11111 << std::endl << std::endl;

	return 0;
}
```

```
     Hello
Hello
Hello

Hello****
Hello**

1.1e+02
1.1
111.11111
1.11111

111.11
1.11
111.1111100000
1.1111100000
```

C++ Operators - Manipulators

```
#include<iostream>

std::ostream & have_fun(std::ostream &output)
{
	output << std::endl << "Have Fun";
	return output;
}

int main()
{
	std::cout << "Hello there,";
	std::cout << have_fun << ", you guys!!" << std::endl;

	std::endl(have_fun(std::cout) << ", you guys!!");

	return 0;
}
```

```
Hello there,
Have Fun, you guys!!

Have Fun, you guys!!
```