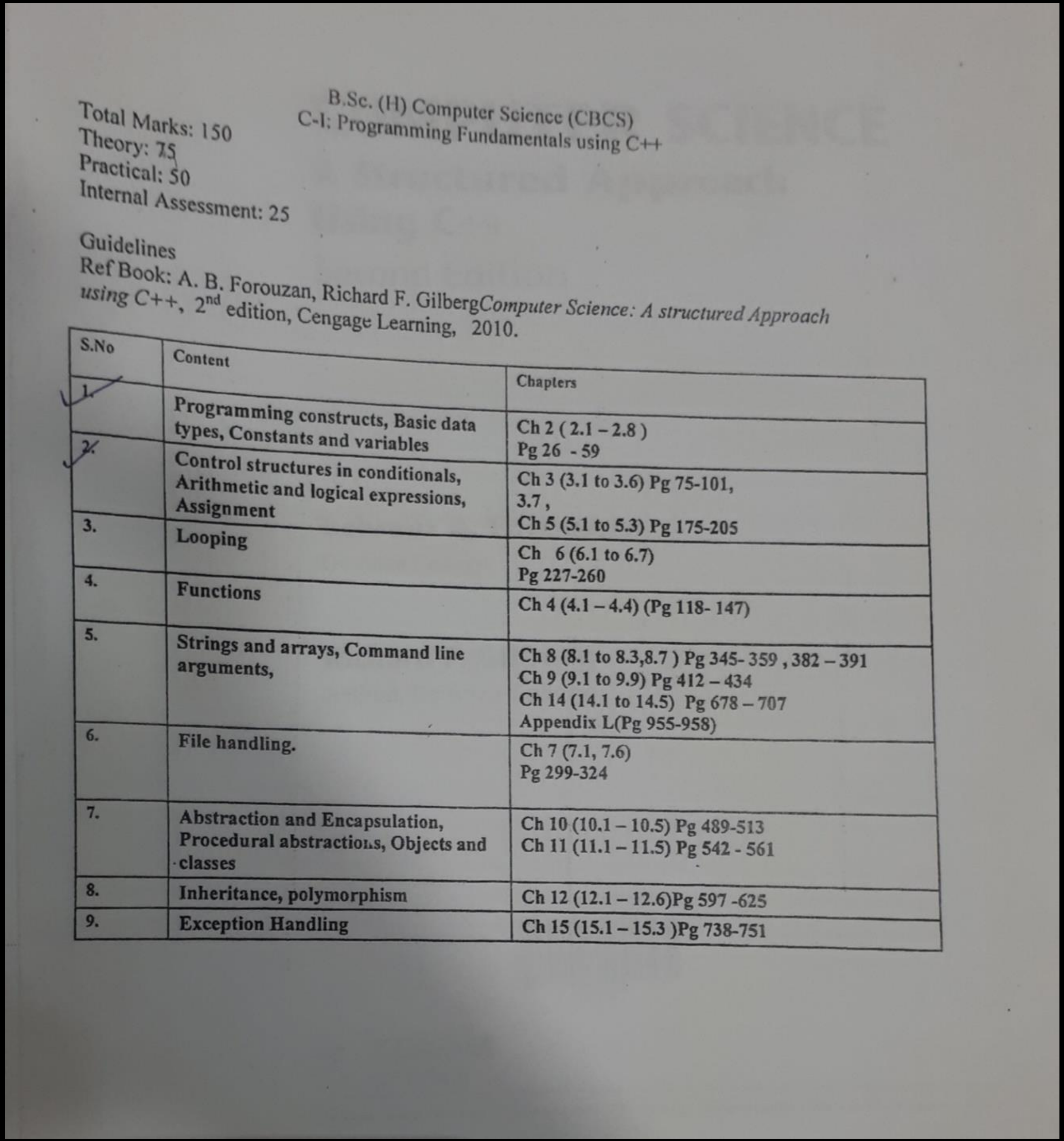
Programming Fundamentals Using C++

Guidelines



B.Sc. (H) Computer Science (CBCS)
C-I: Programming Fundamentals using C++

Total Marks: 150
Theory: 75
Practical: 50
Internal Assessment: 25

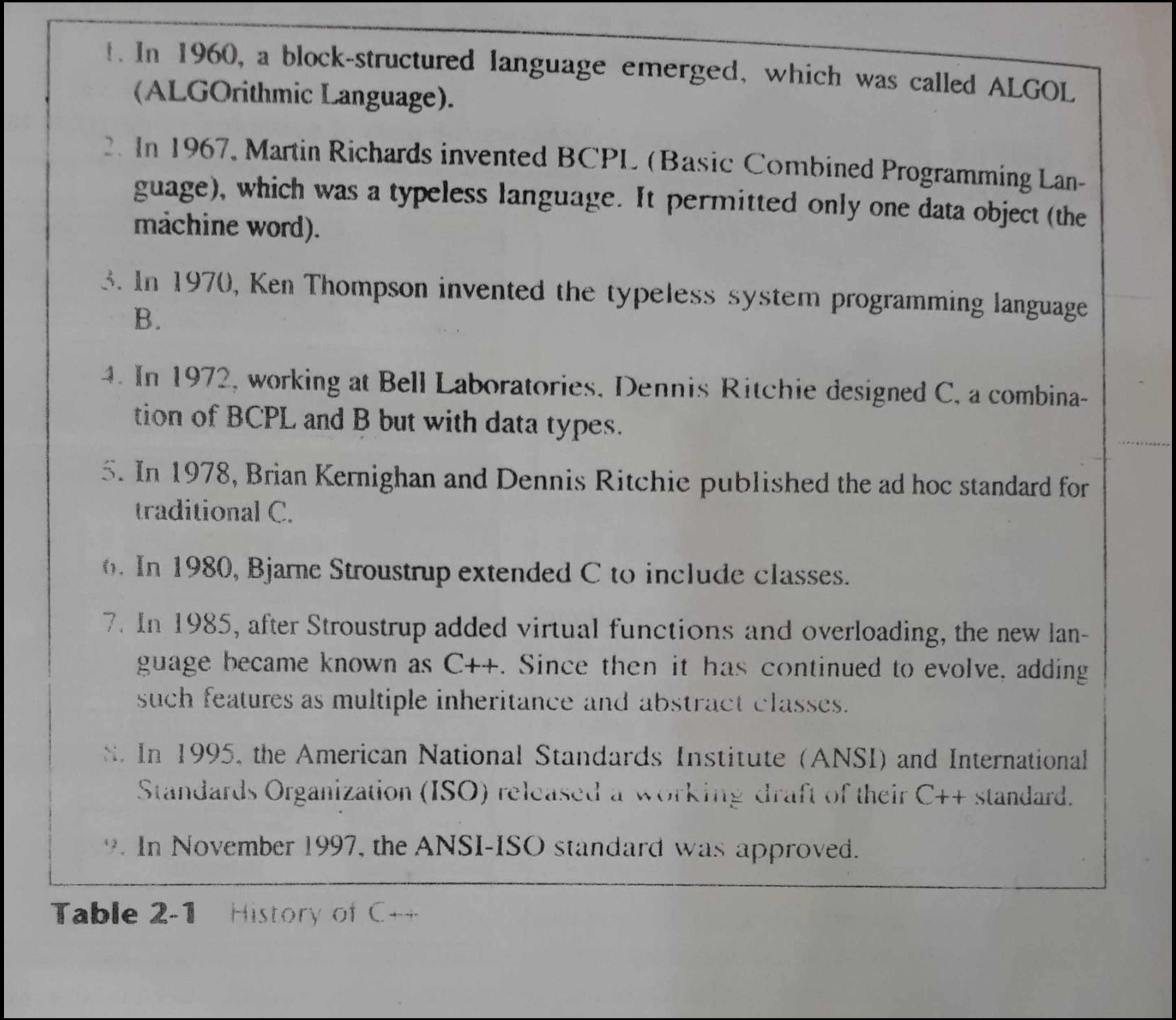
Guidelines
Ref Book: A. B. Forouzan, Richard F. Gilberg*Computer Science: A structured Approach using C++*, 2nd edition, Cengage Learning, 2010.

S.No	Content	Chapters
✓1.	Programming constructs, Basic data types, Constants and variables	Ch 2 (2.1 – 2.8) Pg 26 - 59
✓2.	Control structures in conditionals, Arithmetic and logical expressions, Assignment	Ch 3 (3.1 to 3.6) Pg 75-101, 3.7 , Ch 5 (5.1 to 5.3) Pg 175-205
3.	Looping	Ch 6 (6.1 to 6.7) Pg 227-260
4.	Functions	Ch 4 (4.1 – 4.4) (Pg 118- 147)
5.	Strings and arrays, Command line arguments,	Ch 8 (8.1 to 8.3,8.7) Pg 345- 359 , 382 – 391 Ch 9 (9.1 to 9.9) Pg 412 – 434 Ch 14 (14.1 to 14.5) Pg 678 – 707 Appendix L(Pg 955-958)
6.	File handling.	Ch 7 (7.1, 7.6) Pg 299-324
7.	Abstraction and Encapsulation, Procedural abstractions, Objects and classes	Ch 10 (10.1 – 10.5) Pg 489-513 Ch 11 (11.1 – 11.5) Pg 542 - 561
8.	Inheritance, polymorphism	Ch 12 (12.1 – 12.6)Pg 597 -625
9.	Exception Handling	Ch 15 (15.1 – 15.3)Pg 738-751

C++ is a :

- Structured Programming Language
- High Level Language
- Case-Sensitive (In other words, uppercase and lowercase letters are considered to be different)
- Strongly Typed

2.1 Background



1. In 1960, a block-structured language emerged, which was called ALGOL (ALGOrithmic Language).
2. In 1967, Martin Richards invented BCPL (Basic Combined Programming Language), which was a typeless language. It permitted only one data object (the machine word).
3. In 1970, Ken Thompson invented the typeless system programming language B.
4. In 1972, working at Bell Laboratories, Dennis Ritchie designed C, a combination of BCPL and B but with data types.
5. In 1978, Brian Kernighan and Dennis Ritchie published the ad hoc standard for traditional C.
6. In 1980, Bjarne Stroustrup extended C to include classes.
7. In 1985, after Stroustrup added virtual functions and overloading, the new language became known as C++. Since then it has continued to evolve, adding such features as multiple inheritance and abstract classes.
8. In 1995, the American National Standards Institute (ANSI) and International Standards Organization (ISO) released a working draft of their C++ standard.
9. In November 1997, the ANSI-ISO standard was approved.

Table 2-1 History of C++

2.2 A Sample Program

```
#include <iostream.h>
#include<conio.h>
using namespace std;
int main()
{
cout<<“Hello World”<<endl;
getch();
return 0;
}
```

2.3 Identifiers

- The first character must be alphabetic character or underscore.
- The identifier must consist only of alphabetic characters, digits, and underscores.
- The identifier cannot duplicate a reserved word

2.4 Data Types

- Int (2 Bytes) [short, long, signed, unsigned(all positives)]
- long int(4 bytes)
- By default, int is short(2 bytes)
- Signed Int has it’s range from (-32,768 to 32,767)
- Unsigned Int has it’s range from (0 to 65,535)
- Unsigned long int has it’s range from (0 to 2,147,483,647)
- Signed long int has it’s range from (-2147483648 to 2147483647)

- Char(1 Byte)
- Float(4 Bytes)
- Double(8 Bytes)
- Long double(10 Bytes)
- Bool(1 Byte)
- Void – This type has no values and no operations.

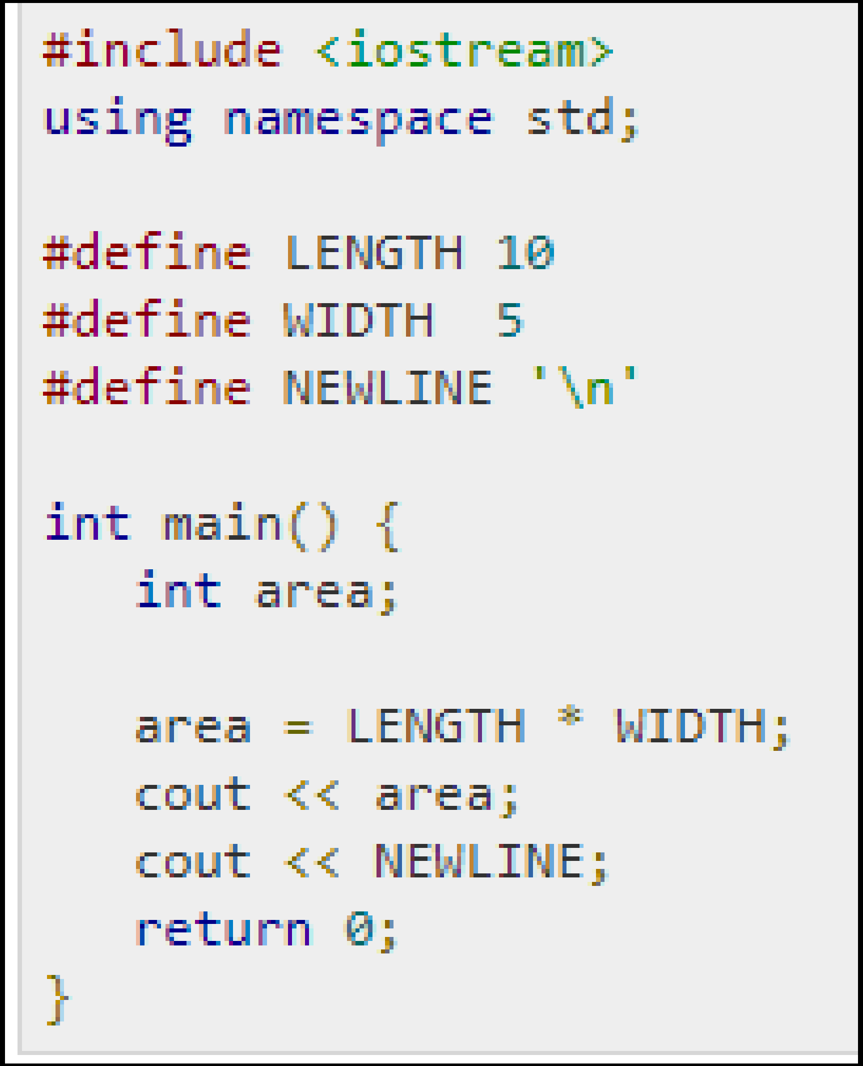
2.5 Variables

- Named memory locations that have a type are called as variables.
- Every variable has a data-type and an identifier(name).
- For e.g. int x;
- Each variable must be declared and defined.
- Declaration is used to name an object, such as a variable.
- Definitions are used to create the object.
- A variable is declared and defined the same time.
- Variable initialization (For e.g. int x=0;)

2.6 Constants/Literals

- Constants are data values that cannot be changed during the execution of a program.
- There are two simple ways in C++ to define constants –
  1. Using **#define** preprocessor/ Macros
  2. Using **const** keyword.
- For e.g. const int length=5;

2.7 Coding Constants



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

#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'

int main() {
   int area;

   area = LENGTH * WIDTH;
   cout << area;
   cout << NEWLINE;
   return 0;
}
```

2.8 Reading and Writing Data

- A stream is an abstract representation of an input data source or output data destination
- C++ automatically defines four standard streams called :
    1. Console input (cin)
    2. Console output(cout)
    3. Console error(cerr)
    4. Console log(clog)
- The difference between console error and console log is that console log is buffered while the console error is unbuffered.
- A buffer is a temporary storage area that holds data while it is being received or accumulated

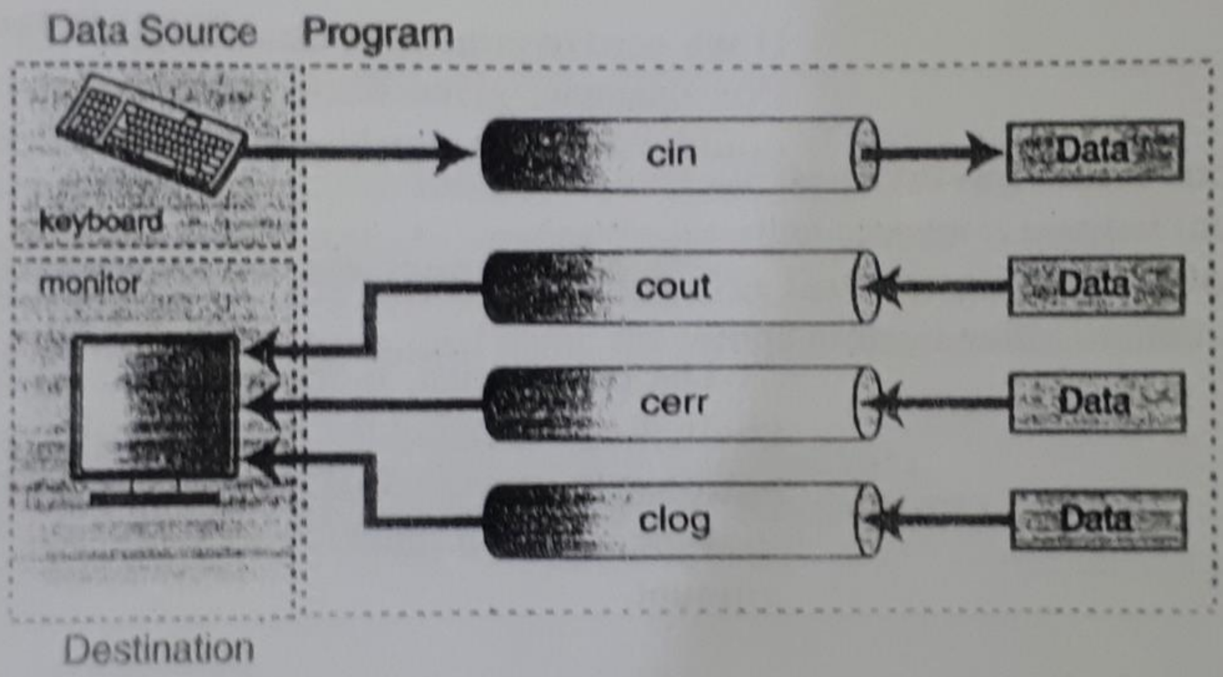
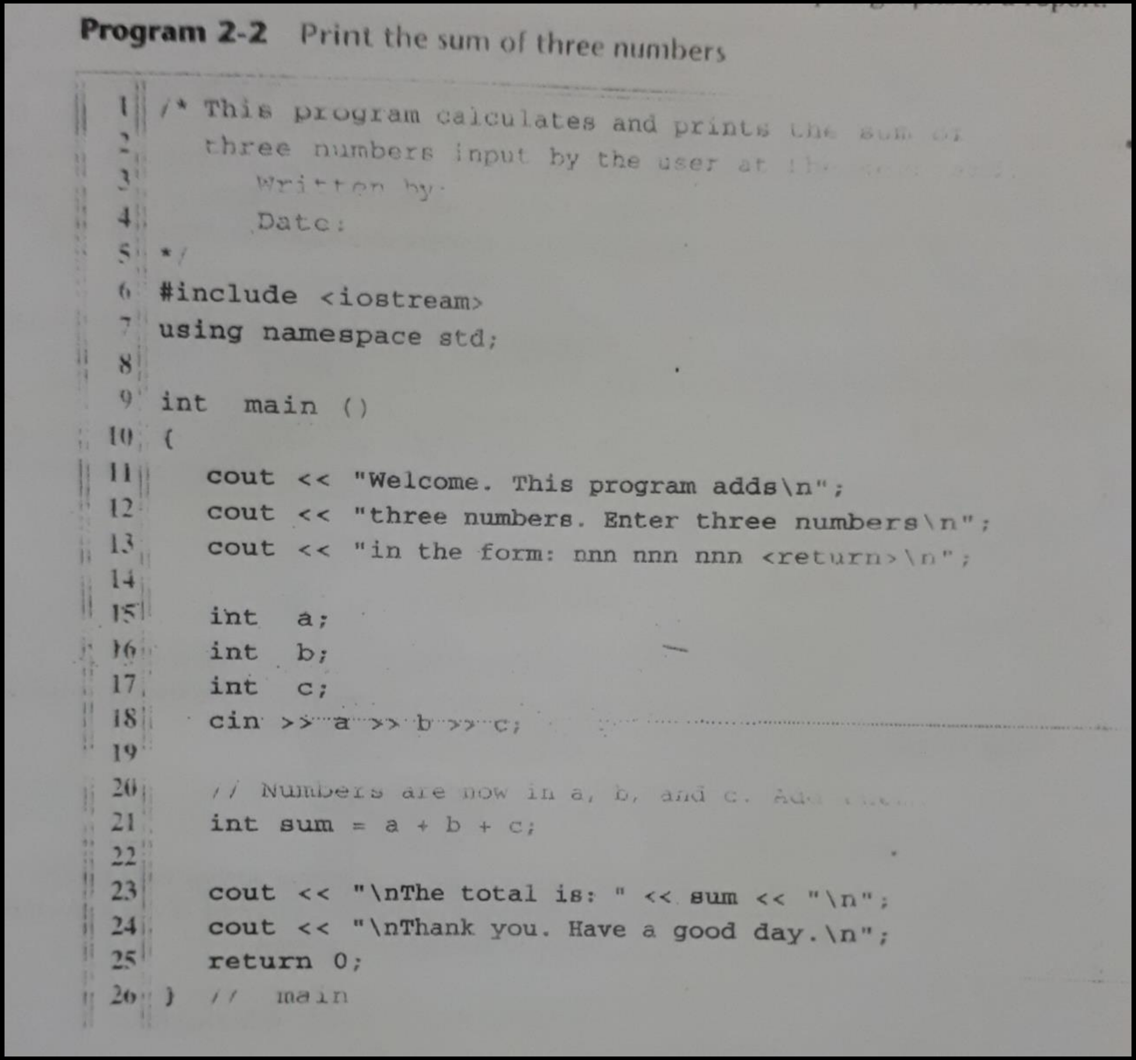
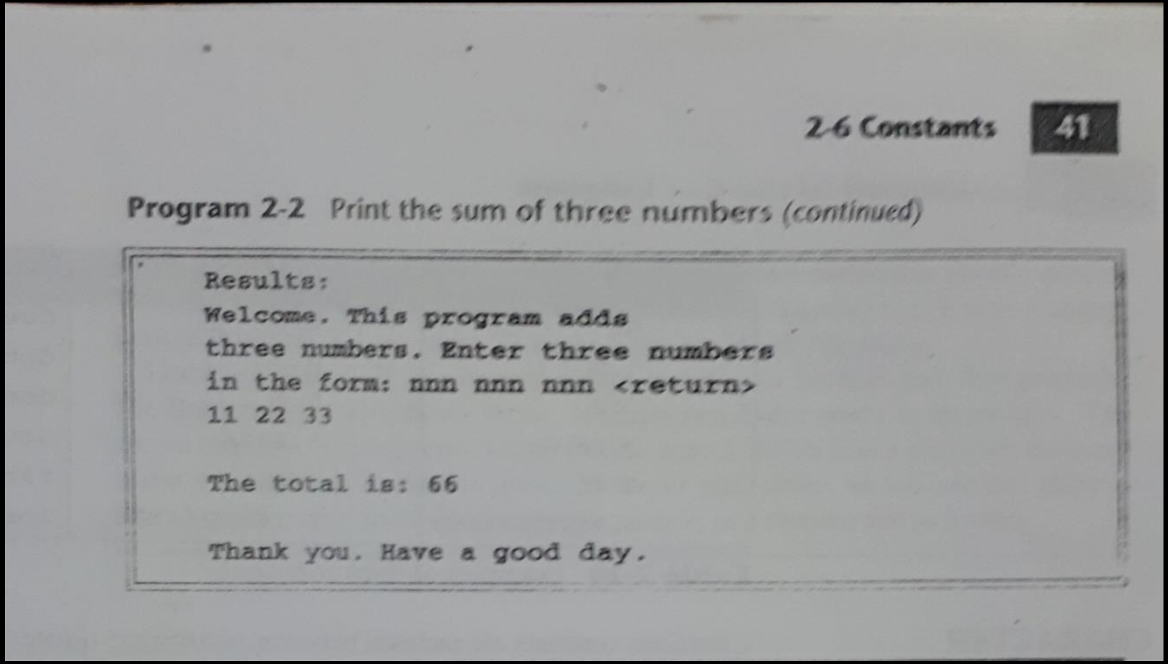


Figure 2-12 Standard streams



Program 2-2 Print the sum of three numbers

```
/* This program calculates and prints the sum of
   three numbers input by the user at the keyboard.
      Written by:
      Date:
*/
#include <iostream>
using namespace std;

int  main ()
{
   cout << "Welcome. This program adds\n";
   cout << "three numbers. Enter three numbers\n";
   cout << "in the form: nnn nnn nnn <return>\n";

   int  a;
   int  b;
   int  c;
   cin >> a >> b >> c;

   // Numbers are now in a, b, and c. Add them.
   int sum = a + b + c;

   cout << "\nThe total is: " << sum << "\n";
   cout << "\nThank you. Have a good day.\n";
   return 0;
}  //  main
```



Program 2-2 Print the sum of three numbers *(continued)*

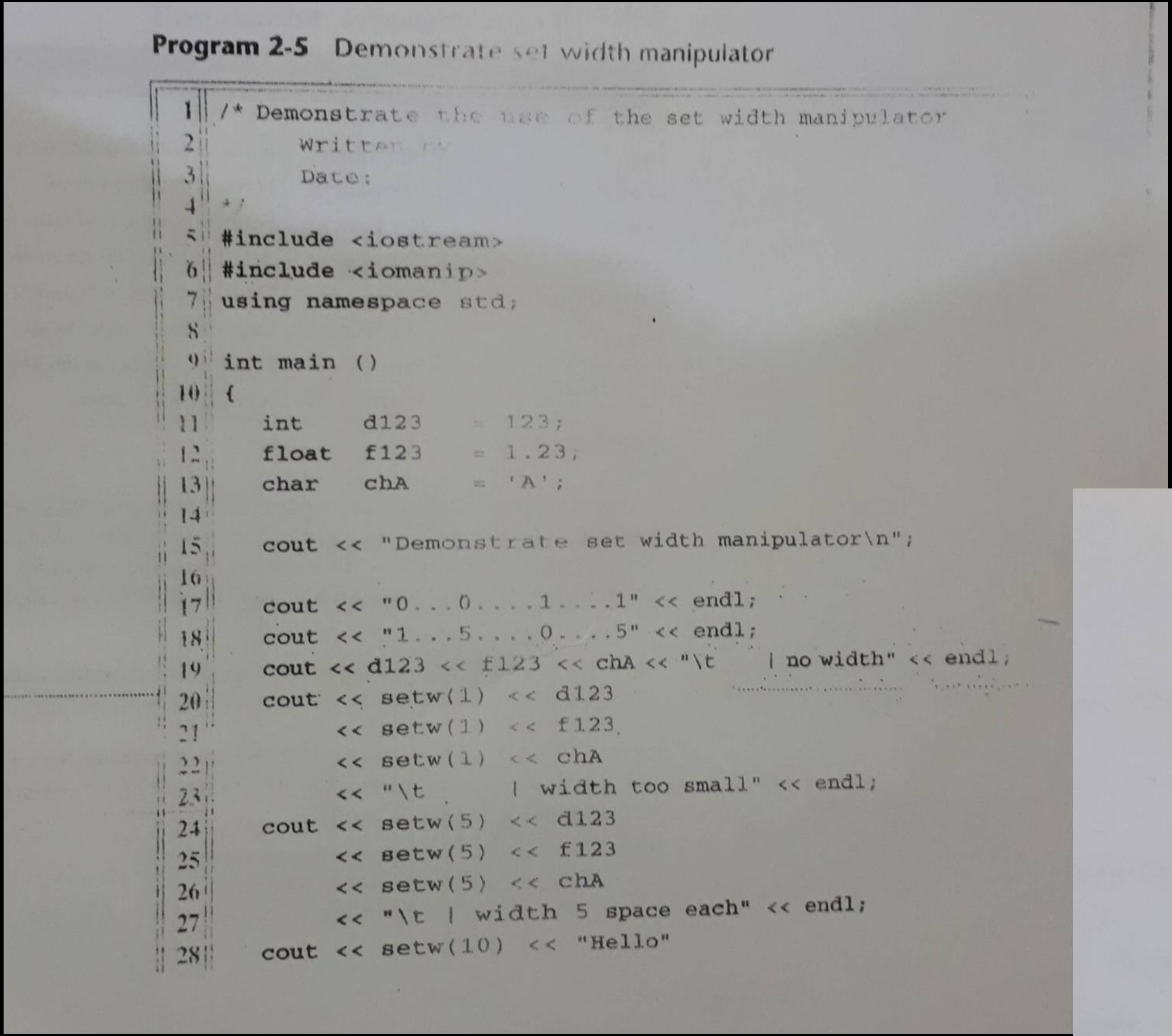
```
Results:
Welcome. This program adds
three numbers. Enter three numbers
in the form: nnn nnn nnn <return>
11 22 33

The total is: 66

Thank you. Have a good day.
```

Program 2-4 Printing different types with cout

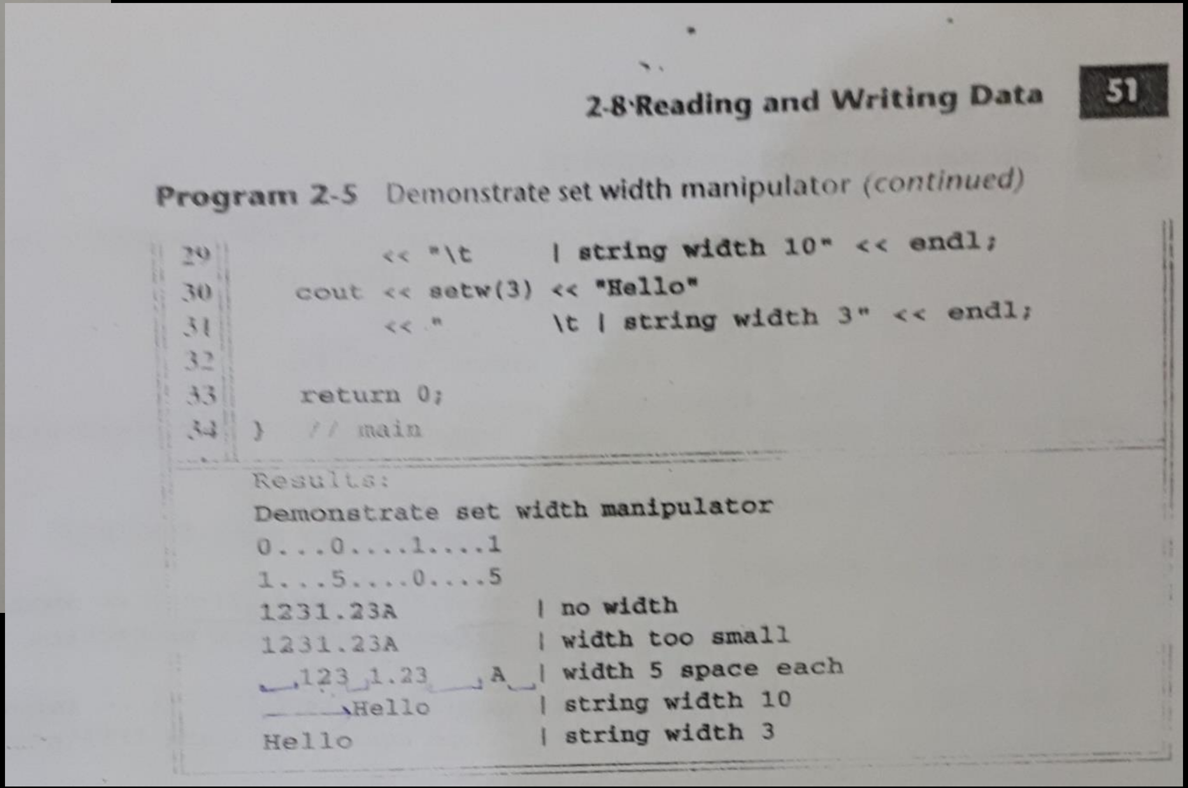
```
/* This program demonstrates the use of the insertion
   operator with several different types
      Written by:
      Date:
*/
#include <iostream>
using namespace std;

int main ()
{
// Statements
   cout << 24      << "\n";
   cout << 12.3    << "\n";
   cout << 'A'     << "\n";
   cout << "Hello" << "\n";
   return 0;
}  // main
```

```
Results:
24
12.3
A
Hello
```

Program 2-4 Analysis In this simple program, each piece of data is displayed on a separate line. To make them come out on separate lines, we chained each data value with the for the newline character ("\n"). Had we not done that, all of the data would have appeared on one line, packed together with no spacing.


Program 2-5 Demonstrate set width manipulator

```
/* Demonstrate the use of the set width manipulator
      Written by:
      Date:
*/
#include <iostream>
#include <iomanip>
using namespace std;

int main ()
{
   int     d123    = 123;
   float   f123    = 1.23;
   char    chA     = 'A';

   cout << "Demonstrate set width manipulator\n";

   cout << "0...0....1....1" << endl;
   cout << "1...5....0....5" << endl;
   cout << d123 << f123 << chA << "\t     | no width" << endl;
   cout << setw(1) << d123
        << setw(1) << f123
        << setw(1) << chA
        << "\t      | width too small" << endl;
   cout << setw(5) << d123
        << setw(5) << f123
        << setw(5) << chA
        << "\t | width 5 space each" << endl;
   cout << setw(10) << "Hello"
```



Program 2-5 Demonstrate set width manipulator *(continued)*

```
        << "\t     | string width 10" << endl;
   cout << setw(3) << "Hello"
        << "      \t | string width 3" << endl;

   return 0;
}  // main
```

```
Results:
Demonstrate set width manipulator
0...0....1....1
1...5....0....5
1231.23A            | no width
1231.23A            | width too small
  123 1.23    A     | width 5 space each
     Hello          | string width 10
Hello               | string width 3
```

Program 2-6 Demonstrate fill character manipulator *(continued)*

```
int main ()
{
   float  amount = 123.45;

   cout << "Demonstrate fill characters\n\n";

   cout << setw(10) << amount
        << "\tAmount with space fill\n";

   cout << setw(10) << setfill('*') << amount
        << "\tAmount with check protection fill\n";

   cout << setw(10) << setfill(' ') << amount
        << "\tand again with space fill\n";

   return 0;
}  // main
```

```
Results:
Demonstrate fill characters

    123.45     Amount with space fill
****123.45     Amount with check protection fill
    123.45     and again with space fill
```

Integer Manipulators

- They are used to change the display format for the integer values.
- The decimal manipulator(dec) is the default, It tells the system to print the value in decimal.
- Octal(oct) tells cout to print the value using the octal numbering system.
- Hexadecimal(hex) tells cout to print in hexadecimal.
- Each of those manipulators sets the printing until it is reset by nother manipulator.

Program 2-7 Demonstrate numeric manipulators *(continued)*

```
int main ()
{
   int    d123    = 123;

   cout << "Values in decimal: \t";
   cout << setw(5) << d123 << setw(5) << d123 << endl;

   cout << "Values in hexadecimal: \t";
   cout << hex;
   cout << setw(5) << d123 << setw(5) << d123 << endl;

   cout << "Values in octal: \t";
   cout << oct;
   cout << setw(5) << d123 << setw(5) << d123 << endl;

   cout << "Values in decimal: \t";
   cout << dec;
   cout << setw(5) << d123 << setw(5) << d123 << endl;

   return 0;
}  // main
```

```
Values in decimal:         123  123
Values in hexadecimal:      7b   7b
Values in octal:           173  173
Values in decimal:         123  123
```

Floating point manipulators

- Fixed - This manipulator tells cout that floating point numbers are to be displayed with fixed-point rather than floating-point numbers.
- Set precision – This manipulator is used to control the number of decimal places to be displayed.
- Show Point – This manipulator displays the value with a decimal point.

```
/* Demonstrate floating-point manipulators.
      Written by.
      Date:
*/
#include <iostream>
#include <iomanip>
using namespace std;

int main ()
{
   cout << "Demonstrate float manipulators\n\n";

   float  f1          = 1.0;
   float  f1234       = 1.234;
   float  f123456789  = 1234567.875;

   cout << f1 << "\t\t\t\tWith no manipulators\n";
   cout << f1234 << endl;
   cout << f123456789 << endl << endl;

   cout << fixed;
   cout << f1 << "\t\tWith fixed added\n";
   cout << f1234 << endl;
   cout << f123456789 << endl << endl;

   cout << setprecision(2);
   cout << f1 << "\t\t\tWith setprecision added\n";
   cout << f1234 << endl;
   cout << f123456789 << endl << endl;

   cout << setprecision(0);
   cout << f1234 << "\t\t\t\tWith setprecision(0)\n";
   cout << setprecision(0) << showpoint
        << f1234 << "\t\t\t\tWith showpoint\n";
} // main
```

Program 2-8 Demonstrate fixed-point manipulators *(continued)*

```
1                  With no manipulators
1.234
1.23457e+06

1.000000           With fixed added
1.234000
1234567.875000

1.00               With setprecision added
1.23
1234567.88

1                  With setprecision(0)
1.                 With showpoint
```

```
int main ()
{
   int a = 3;
   int b = 4;
   int c = 5;
   int x;
   int y;

   cout << "Initial values of the variables: \n";
   cout << "a = " << a << "   b = " << b << "   c = "
        << c << endl;
   cout << endl;

   x = a * 4 + b / 2 - c * b;
   cout << "Value of a * 4 + b / 2 - c * b is: "
        << x << endl;

   y = --a * (3 + b) / 2 - c++ * b;
   cout << "Value of --a * (3 + b) / 2 - c++ * b is: "
        << y << endl;

   cout << "\nValues of the variables are now: \n";
   cout << "a = " << a << "   b = " << b << "   c = "
        << c << endl;

   return 0;
}  // main
```

*4

```
Results:
Initial values of the variables:
a = 3   b = 4   c = 5

Value of a * 4 + b / 2 - c * b is: -6
Value of --a * (3 + b) / 2 - c++ * b is: -13

Values of the variables are now:
a = 2   b = 4   c = 6
```

A warning is in order: In C++, if an expression variable is modified more than once in

```cpp
/* Demonstrate implicit casts of numeric types.
      Written by:
      Date:
*/
#include <iostream>
using namespace std;

int main ()
{
   char   aChar    = 'A';
   int    printChar;
   int    intNum   = 200;
   double fltNum   = 245.3;

   cout << "aChar contains :  " << aChar  << endl;
   printChar = aChar;
   cout << "aChar numeric  :  " << printChar  << endl;
   cout << "intNum contains:  " << intNum << endl;
   cout << "fltNum contains:  " << fltNum << endl;

   intNum = intNum + aChar;   // aChar converted to int
   fltNum = fltNum + aChar;  // aChar converted to float

   cout << "\nAfter additions...\n";
   printChar = aChar;
   cout << "aChar numeric  :  " << printChar  << endl;
   cout << "intNum contains:  " << intNum     << endl;
   cout << "fltNum contains:  " << fltNum     << endl;
   return 0;
}  // main
```

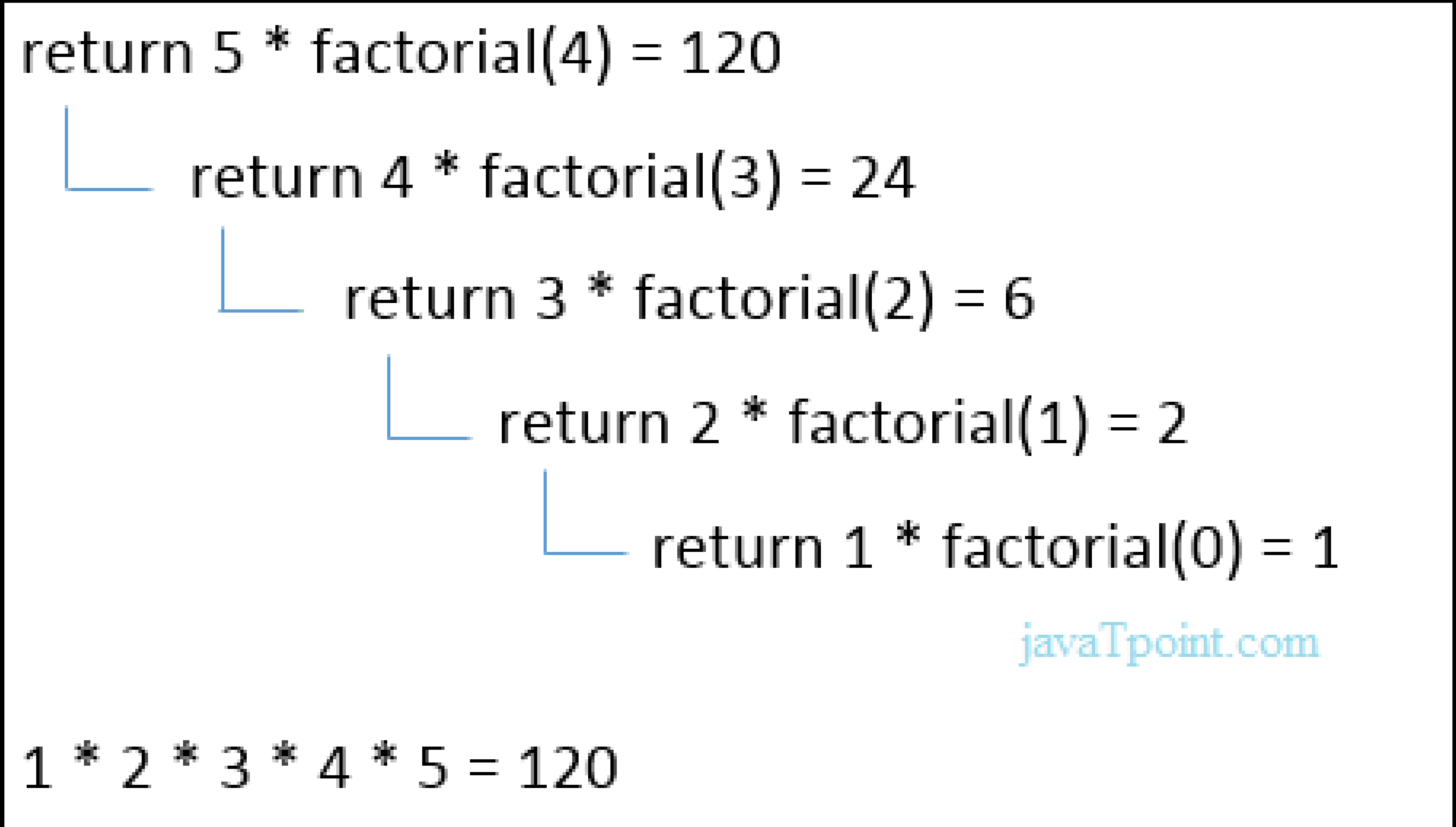
```
Results:
aChar contains :  A ✓
aChar numeric  :  65 ✓
intNum contains:  200 ✓
fltNum contains:  245.3 ✓

After additions...
aChar numeric  :  65 ✓
intNum contains:  265 ✓
fltNum contains:  310.3 ✓
```

Recursion

- When function is called within the same function, it is known as recursion in C++.
- The function which calls the same function, is known as recursive function.

```
int factorial( int N )
{
  int product = 1;
  for ( int j=1; j<=N; j++ )
    product =product * j;
  return product;
}
int main()
{
int num;
cout<<“Enter the number”;
cin>>num;
cout<<“Factorial of that number is”<<factorial(num);
return 0;
}
```

```
int factorial(int n)
{
    if (n == 0)
      return 1;
    return (n*factorial(n-1));
}
int main()
{
int num;
cout<<“Enter the number”;
cin>>num;
cout<<“Factorial of that number is”<<factorial(num);
return 0;
}
```



```
return 5 * factorial(4) = 120
    return 4 * factorial(3) = 24
        return 3 * factorial(2) = 6
            return 2 * factorial(1) = 2
                return 1 * factorial(0) = 1
```

1 * 2 * 3 * 4 * 5 = 120

Fig: Recursion

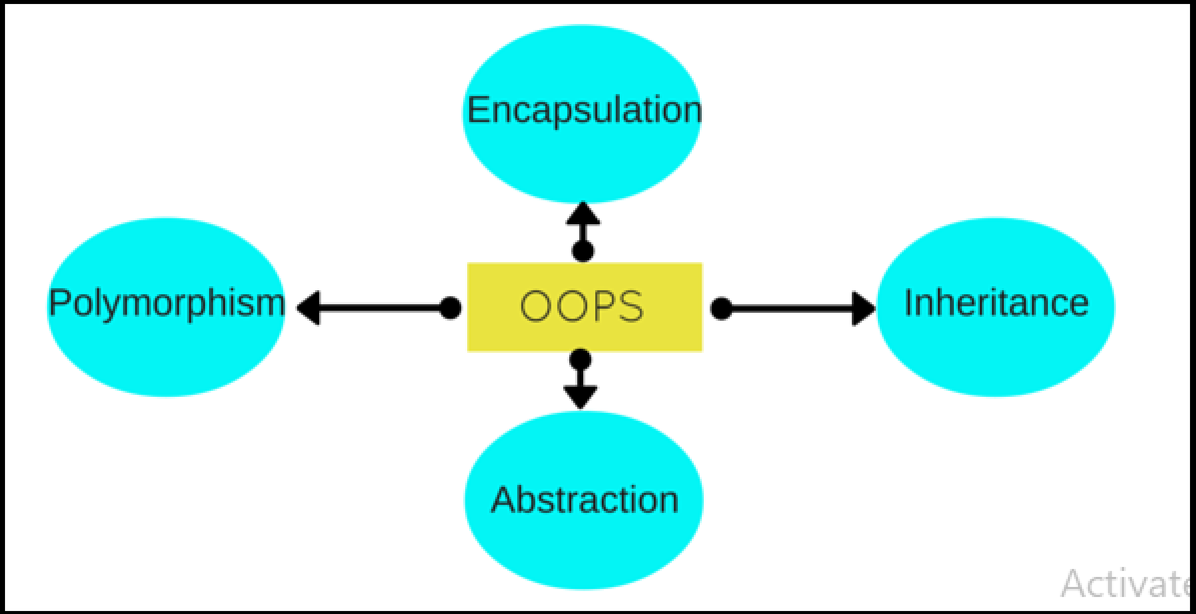
Templates in C++

- Template is simple and yet very powerful tool in C++.
- The simple idea is to pass data type as a parameter so that we don’t need to write same code for different data types.
- " **typename** " is a keyword in the C++ programming language used when writing **templates**. It is used for specifying that a dependent name in a **template** definition or declaration is a type.

```
#include <iostream.h>

// One function works for all data types.

template <typename T>
T myMax(T x, T y)
{
   return (x > y)? x: y;
}
int main()
{
  cout << myMax<int>(3, 7) << endl;  // Call myMax for int
  cout << myMax<double>(3.0, 7.0) << endl; // call myMax for double
  cout << myMax<char>('g', 'e') << endl;   // call myMax for char

  return 0;
}
```

OOPs (Object Oriented Programming)

- Object Oriented programming is a programming style that is associated with the concept of Class, Objects and various other concepts revolving around these two, like Inheritance, Polymorphism, Abstraction, Encapsulation etc.



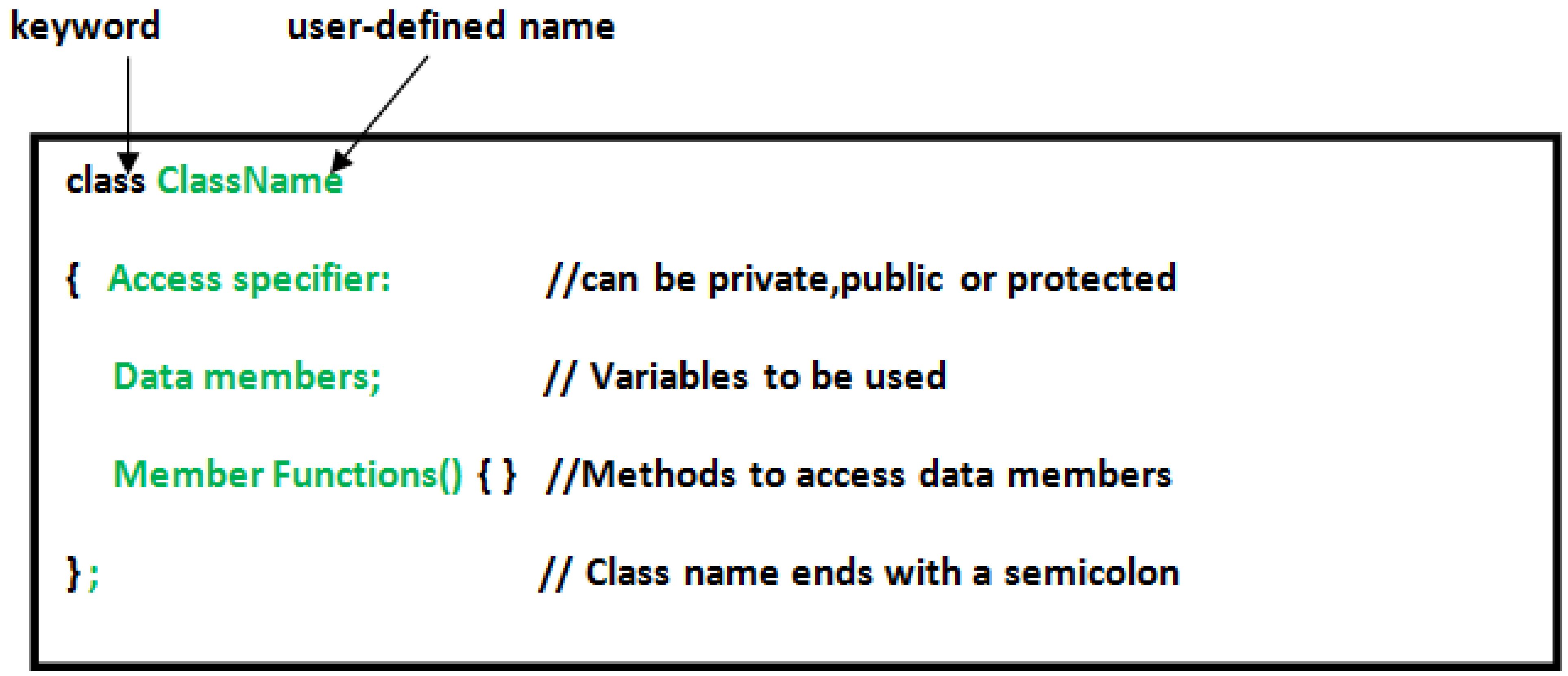
- **Data abstraction** refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.
- **Encapsulation** is placing the data and the functions that work on that data in the same place.
- One of the most useful aspects of object-oriented programming is code reusability. As the name suggests **Inheritance** is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class.
- Poly refers to many. That is a single function or an operator functioning in many ways different upon the usage is called **polymorphism**.

C++ Classes and Objects

- The building block of C++ that leads to Object Oriented programming is a **Class**.
- It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.
- An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Defining Class and Declaring Objects

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.



```
class ClassName

{  Access specifier:             //can be private,public or protected

   Data members;                 // Variables to be used

   Member Functions() { }        //Methods to access data members

};                               // Class name ends with a semicolon
```

```cpp
using namespace std;
class Geeks
{
   // Access specifier
   public:

   // Data Members
   string geekname;

   // Member Functions()
   void printname()
   {
      cout << "Geekname is: " << geekname;
   }
};
```

```cpp
int main() {

   // Declare an object of class
geeks
   Geeks obj1;

   // accessing data member
   obj1.geekname = "Abhi";

   // accessing member
function
   obj1.printname();
   return 0;
}
```

Member Functions in Classes

There are 2 ways to define a member function:

- Inside class definition
- Outside class definition

To define a member function outside the class definition we have to use the scope resolution :: operator along with class name and function name

```
using namespace std;
class Geeks
{
    public:
    string geekname;
    int id;

    // printname is not defined inside class defination
    void printname();

    // printid is defined inside class defination
    void printid()
    {
        cout << "Geek id is: " << id;
    }
};
```

```
// Definition of printname using scope resolution operator ::
void Geeks::printname()
{
    cout << "Geekname is: " << geekname;
}
int main() {

    Geeks obj1;
    obj1.geekname = "xyz";
    obj1.id=15;

    // call printname()
    obj1.printname();
    cout << endl;

    // call printid()
    obj1.printid();
    return 0; }
```

Constructors

- Constructors are special class members which are called by the compiler every time an object of that class is instantiated. Constructors have the same name as the class and may be defined inside or outside the class definition.

There are 3 types of constructors:

- Default constructors
- Parametrized constructors
- Copy constructors

```
using namespace std;
class Geeks
{
    public:
    int id;

    //Default Constructor
    Geeks()
    {
        cout << "Default Constructor called" << endl;
        id=-1;
    }

    //Parametrized Constructor
    Geeks(int x)
    {
        cout << "Parametrized Constructor called" << endl;
        id=x;
    }
};
```

```
int main() {

    // obj1 will call Default Constructor
    Geeks obj1;
    cout << "Geek id is: " <<obj1.id << endl;

    // obj1 will call Parametrized Constructor
    Geeks obj2(21);
    cout << "Geek id is: " <<obj2.id << endl;
    return 0;
}
```

```
Default Constructor called
Geek id is: -1
Parametrized Constructor called
Geek id is: 21
```

Copy Constructor

- A Copy Constructor creates a new object, which is exact copy of the existing copy. The compiler provides a default Copy Constructor to all the classes.
- Syntax:

```
class-name (class-name &)
 {
 }
```

Destructors

- Destructor is another special member function that is called by the compiler when the scope of the object ends.
- Destructors have same name as the class preceded by a tilde (~).
- There can only one destructor in a class with classname preceded by ~, no parameters and no return type.
- Destructors don’t take any argument and don’t return anything

2	**Class Access Modifiers** A class member can be defined as public, private or protected. By default members would be assumed as private.
3	**Constructor & Destructor** A class constructor is a special function in a class that is called when a new object of the class is created. A destructor is also a special function which is called when created object is deleted.
4	**Copy Constructor** The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously.
5	**Friend Functions** A **friend** function is permitted full access to private and protected members of a class.
6	**Inline Functions** With an inline function, the compiler tries to expand the code in the body of the function in place of a call to the function.