OBJECT:

- To understand some library functions.
- To demonstrate setw() manipulator
- Data type automatic conversion and casting.
- The bool Data Type

LIBRARY FUNCTIONS

Library functions which are also called as "built-in" functions are already available and implemented in C++. We can directly call these in our program any time. Library functions in C++ are declared and defined in special files called "Header Files" which we can reference in our C++ programs using the "include" directive. For Example, to include all the built-in functions related to math, we should include <cmath> header as follows:

```
#include <cmath>
```

HEADER FILES

As with cout and other such objects, you must #include a header file that contains the declaration of any library functions you use. To use the sqrt() function, the required header file is CMATH. If you don't include the appropriate header file when you use a library function, you'll get an error message.

LIBRARY FILES

Files containing library functions and objects will be linked to your program to create an executable file. These files contain the actual machine-executable code for the functions. Such library files often have the extension ".LIB". The sqrt() function is found in such a file. It is automatically extracted from the file by the linker, and the proper connections are made so that it can be called (that is, invoked or accessed). Your compiler takes care of all these details for you, so ordinarily you do not need to worry about the process.

HEADER FILES AND LIBRARY FILES

To use a library function like sqrt(), you must link the library file that contains it to your program. The appropriate functions from the library file are then connected to your program by the linker. The functions in your source file need to know the names and types of the functions and other elements in the library file. They are given this information in a header file. Each header file contains information for a particular group of functions. The functions themselves are grouped together in a library file, but the information about them is scattered throughout a number of header files.

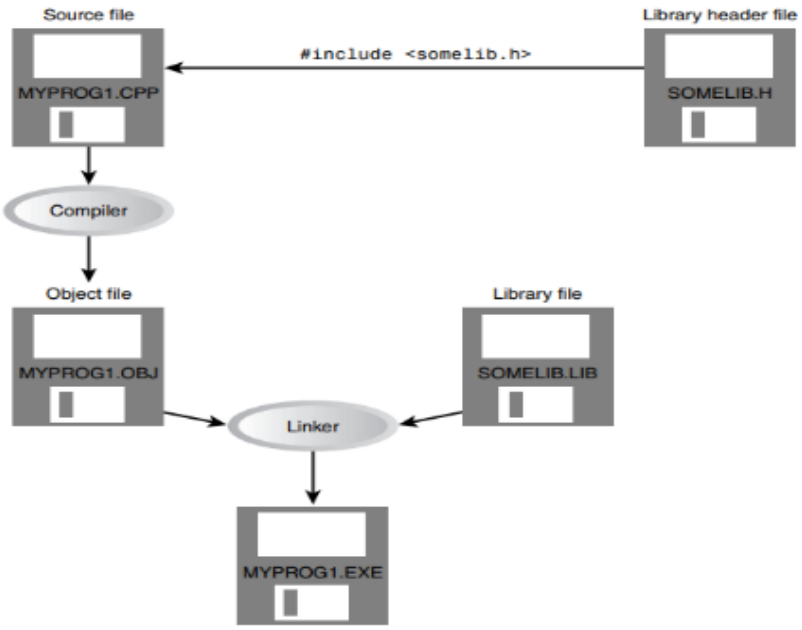


Figure 1: Header files and library Files

PROGRAM 1: Demonstrating the library function sqrt().

```
#include <iostream> //for cout, etc.
#include <cmath> //for sqrt()

using namespace std;
int main()
{
double number, answer; //sqrt() requires type double
cout << "Enter a number: ";
cin >> number; //get the number

answer = sqrt(number); //find square root
cout << "Square root is " << answer << endl; //display it
return 0;
}                 //end function body
```

PROGRAM 2: Demonstrating the library function pow().

```
#include <iostream> //for cout, etc.
#include <cmath> //for pow()

using namespace std;
int main()
{
int num, power;
double ans;
cout << "Enter a number: ";
cin >> num;                //get the number
cout<<"Enter power";
cin>>power;
ans = pow(num, power);
cout << num << " power " << power << " = " << ans << endl;
return 0;
}
```

THE SETW() MANIPULATOR

The setw manipulator causes the number (or string) that follows it in the stream to be printed within a field n characters wide, where n is the argument to setw(n). The value is right justified within the field. The declarations for the manipulators (except endl) are not in the usual IOSTREAM header file, but in a separate header file called IOMANIP. When you use these manipulators you must #include this header file in your program.

PROGRAM 3: Demonstrating the setw() Manipulator.

```
#include <iostream>
#include <iomanip> // for setw
using namespace std;
int main(){
int m1=512, m2=612, m3=92;
cout << setw(15) << "NAME" << setw(8) << "MARKS" << endl
     << setw(15) << "Ali Ahmed" << setw(8) << m1 << endl
     << setw(15) << "Nabeel Arif" << setw(8) << m2 << endl
     << setw(15) << "Adeel Rasheed" << setw(8) << m3 << endl;
return 0;
}
```

CASTING

When the user manually changes data from one type to another, this is known as **explicit conversion**. This type of conversion is also known as **type casting**. Type casting is a way to convert a variable from one data type to another data type. For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'. You can convert the values from one type to another explicitly.

There are several kinds of casts in Standard C++: static casts, dynamic casts, reinterpret casts, and const casts. Here we'll be concerned only with static casts.

There are three major ways in which we can use explicit conversion in C++. They are:

- C-style type casting (also known as **cast notation**)
- Function notation (also known as **old C++ style type casting**)
- Type conversion operators

C-STYLE TYPE CASTING

As the name suggests, this type of casting is favored by the **C programming language**. It is also known as **cast notation**.

The syntax for this style is:

```
(type_name) expression
```

FUNCTION-STYLE CASTING

We can also use the function like notation to cast data from one type to another.

The syntax for this style is:

```
type_name(expression);
```

TYPE CONVERSION OPERATORS

Besides these two type castings, C++ also has four operators for type conversion. They are known as **type conversion operators**. They are:

Here's a statement that uses a C++ cast to change a variable of type int into a variable of type char:

```
aCharVar = static_cast<char>(anIntVar);
```

PROGRAM 4: Demonstrating the explicit casting.

```
#include <iostream>

using namespace std;

int main() {

    // initializing a double variable
    double d = 3.56;
    cout << "d = " << d << endl;

    // C-style conversion from double to int
    int a = (int) d;
    cout << "a   = " << a << endl;

    // function-style conversion from double to int
    int b = int(d);
    cout << "b   = " << b << endl;

   // Type conversion operators from double to int
    int c = static_cast<int>(d);
    cout << "c   = " << c << endl;

    return 0;
}
```

AUTOMATIC CONVERSION

Type conversions can be implicit which is performed by the compiler automatically, When two operands of different types are encountered in the same expression, the lower-type variable is converted to the type of the higher-type variable. Types are considered "higher" or "lower," based roughly on the order shown below:

Highest **Lowest**

long double ← Double ← Float ← long ← int ← short ← char

The problem we had in the previous exercise was due to the Automatic Conversion. Consider the following code:

```
int a = 11;
int b = 2;
float ans = a / b;
```

The value in ans we were getting was 5 instead of 5.5

This is because when 2 int data types are involved in a expression the result in automatically converted in to int which in the only datatype in the expression.

The problem can be solved by casting one of the data type into float by using the statement like this:

```
float ans = (float)a / b;
```

PROGRAM 5: Demonstrating the automatic conversion of data type in expressions.

```
#include <iostream>

using namespace std;

int main() {
    int a = 10;
    int b = 3;

    //Both variable are int the answer in converted to int
     cout<< "a/b="<<a/b<<endl;

     int x = 10;
     float y = 3;

     //One of the variable is float the answer in converted to float
     cout<< "x/y="<<x/y;
     return 0;
}
```

PROGRAM 6: Demonstrating the automatic conversion of data type in expressions.

```
#include <iostream>

using namespace std;

int main() {

     /*Wrong answer due to automatic type conversion
     both int so the final answer will be int */

     cout<<"2/3="<<2/3<<endl;

     /*Still Wrong as the casting is applied
     on answer of expression not on single variable */

     cout<<"2/3="<<(float)(2/3)<<endl;

     /*Correct it by using proper way of casting
     now only 2 is converted into float
     and as one variable in float the answer will
     automatically converted into float*/

     cout<<"2/3="<<(float)2 /3 <<endl;

     return 0;
}
```

BOOL DATA TYPE

Variables of type bool can have only two possible values: true and false. In theory a bool type requires only one bit (not byte) of storage, but in practice compilers often store them as bytes because a byte can be quickly accessed, while an individual bit must be extracted from a byte, which requires additional time.
As we'll see, type bool is most commonly used to hold the results of comparisons. Is alpha less than beta? If so, a bool value is given the value true; if not, it's given the value false.

DECLARATION

```
bool var = true;
```

Printing the bool value on screen will result in 0 for false and 1 for true. For Example:

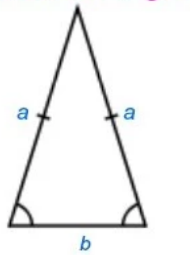
```
Cout<<var;
```

This will give the output:

```
1
```

EXERCISE 1:

An isosceles triangle is a triangle that has two sides of equal length.



$$\text{Area of isosceles triangle} = \frac{1}{2} \times \text{Base} \times \text{Height}$$

$$0.5 \times b \times \sqrt{a^2 - \frac{b^2}{4}}$$

- Write a program which has two variable **a** and **b,** where **a** is the value of two equal sides of triangle and **b** is the value of base. You can use float data type for variable **a** and **b**.
- Now create another variable **area** with double data type.
- Now take the input of variable **a** and **b** from user by printing some proper messages and using cin>>. Also warn user not to give zero or negative value in those messages.
- Now by using **sqrt()** function and **pow()** function calculate the Area of Isosceles triangle by using the above given formula. In last print the value of area which is calculated by the program.
- For Example, sif user give a=3 and b=4 then Area=4.47214

EXERCISE 2:

Write a program which takes the radius of a circle and calculates the area and circumference of the circle. Use pre define **PI** constant from cmath library. Find the formula of both radius and area from internet or any book.

```
Input the radius(1/2 of diameter) of a circle : 5
The area of the circle is : 78.5397
The circumference of the circle is : 31.4159
```