



Introduction to Functions

C++' Names

A function is an assignment or a task you are asking C++ to perform for the functionality of your program. There are two kinds of functions: those supplied to you and those you will be writing. The functions that are supplied to you are usually in three categories: those built-in Microsoft Windows operating system, those written in C++ (they are part of the C++ language), and those written by Borland (they are supplied to you with the compiler, included in the various libraries that are installed with the compiler). The use of these functions is the same regardless of the means you get them; you should know what a function looks like and how to use one, what functions are already available, where they are located, and what a particular function does, how and when to use them.

Function Definition

In order to create and use your function, you must let the compiler know. Letting the compiler know about your function means you "declare" it. The syntax of declaring a function is:

```
ReturnType FunctionName (Needs);
```

In English, an assignment considered a function is made of three parts: its purpose, its needs, and the expectation.

Return Value

The purpose of a function identifies what the function is meant to do. When a function has carried its assignment, it provides a result. For example, if a function were supposed to calculate the area of a square, the result would be the area of a square. The result of a function used to get a student's first name would be a word representing a student's first name.

When you expect a specific result from a function, such as the area of a square, the result would be a numeric value; another kind of result could be a word, etc. The result of a function is called a return value. A function is also said to return a value.

There are two forms of expectations you will have from a function: to return a specific value or to perform a simple assignment. If you want the function to perform an assignment without giving you back a result, such a function is qualified as void and would be declared as

```
void FunctionName(Needs);
```

A return value, if not void, can be any of the data types we have studied so far. This means that a function can return a char, an int, a float, a double, a bool, or a string. Here are examples of declaring functions by defining their return values:

```
double FunctionName(Needs);
char FunctionName(Needs);
bool FunctionName(Needs);
string FunctionName(Needs);
```

Function Names

A function name follows the same rules we have applied to our variables so far. In addition, use a name that specifies what the function is expected to do. Usually, a verb is appropriate for a function that performs an action; an example would be add, start, assign, play, etc.

The names of most functions in C++ Builder start in uppercase. We will follow the same convention in this book. Therefore, the above names would be: Add, Start, Assign, Play.


If the assignment of a function is a combination of words, such as converting a temperature from Celsius to Fahrenheit, start the name of the function with a verb and append the necessary words each starting in uppercase (remember that the name of a function is in one word). Examples include ConvertToFahrenheit, CalculateArea, LoadFromFile, etc. Some functions will not include a verb. They can simply represent a word such as Width, Index, New. They can also be a combination of words; examples include DefaultName, BeforeConstruction, or MethodOfAssignment. Here are examples of function names

```
double CalculateArea(Needs);
char Answer(Needs);
bool InTheBox(Needs);
string StudentName(Needs);
```


Arguments – Parameters

In order to carry its assignment, a function might be supplied something. For example, when a function is used to calculate the area of a square, you have to supply the side of the square, then the function will work from there. On the other hand, a function used to get a student's first name does not have a need; its job is to supply or return something.

Some functions have needs and some don't. The needs of a function are provided between parentheses. These needs could be as varied as possible. If a function does not have a need, leave its parentheses empty.

 In some references, instead of leaving the parentheses empty, the programmer would write void. In this book, if a function does not have a need, we will leave its parentheses empty.

Some functions will have only one need, some others will have many. A function's need is called an argument. If a function has a lot of needs, these are its arguments.

 In some documents, an argument is called a parameter. Both word mean the same thing.

The argument is a valid variable and is defined by its data type and a name. For example, if a function is supposed to calculate the area of a square and it is expecting to receive the side of the square, you can declare it as

```
double CalculateArea(double Side);
```

A function used to get a student's first name could be declared as:

```
string FirstName();
```

One of the biggest questions is to know when to supply arguments or not; if a function will need arguments, how many will be necessary? If a function were

supposed to simply request something from the user, such a function might not need an argument. If a function is supposed to only display a message to the user, it also might not need an argument. On the other hand, a function used to calculate the perimeter of a rectangle would need to know the length and width of the figure. Therefore, the length and the width would be provided as arguments. Here are examples of declaring functions: some take arguments, some don't:

```
double CalculateArea(double Side);
char Answer();
void Message(float Distance);
bool InTheBox(char Mine);
string StudentName();
double RectangleArea(double Length, double Width);
void DefaultBehavior(int Key, double Area, char MI, float Ter);
```

Defining Functions

In order to use a function in your program, you have to let the compiler know what the function does; in fact, the compiler will help you make sure the function can perform its intended assignment. Sometimes (depending on where the function is located in your program), you will not have to declare the function before using it; but you must always tell the compiler what behavior you are expecting.

We have seen that the syntax of declaring a function was:

```
ReturnType FunctionName(Needs);
```

To let the compiler know what the function is meant to do, you have to "define" it. Defining a function means describing its behavior. The syntax of defining a function is:

```
ReturnType FunctionName(Needs) {Body}
```

You define a function using the rule we applied with the main() function. Define it starting with its return value (if none, use void), followed by the function name, its argument (if any) between parentheses, and the body of the function. Once you have defined a function, other functions can use it.

Function Body

As an assignment, a function has a body. The body of the function describes what the function is supposed to do. The body starts with an opening curly bracket "{" and ends with a closing curly bracket "}". Everything between these two symbols belongs to the function. From what we have learned so far, examples of functions would be:

```
double CalculateArea(double Side) {};
char Answer() {};
```

The most used function in C++ is called main().

In the body of the function, you describe the assignment the function is supposed to perform. As simple as it looks, a function can be used to display a message. Here is an example:

```
void Message()
{
    cout << "This is C++ in its truest form.";
}
```

A function can also implement a complete behavior. For example, on a program used to perform geometric shape calculations, you can use different functions to handle specific tasks. Imagine you want to calculate the area of a square. You can

define a particular function that would request the side of the square:

```
cout << "Enter the side of the square: ";
cin >> Side;
```

and let the function calculate the area using the formula $\text{Area} = \text{Side} * \text{Side}$. Here is an example of such a function:

```
void SquareArea()
{
    double Side;

    cout << "\nEnter the side of the square: ";
    cin >> Side;

    cout << "\nSquare characteristics:";
    cout << "\nSide = " << Side;
    cout << "\nArea = " << Side * Side;
}
```

To create a more and effective program, divide jobs among functions and give each function only the necessary behavior and a specific assignment. A good program is not proven by long and arduous functions.

◆ Defining a Function

1. Create a new C++ Console Application based on the Console Wizard
2. Save the project in a folder called Function1. Accept the suggested names of the unit and the project.
3. Change the content of the program as follows: Here is what the Message() function would look like in a program:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    void Message();

    cout << "\nPress any key to continue...";
    getch();
    return 0;
}
//-----
void Message()
{
    cout << "Welcome to the Red Oak High School.";
}
//-----
```

4. To test the program, press F9.

Calling Functions

One of the main reasons of using various functions in your program is to isolate assignments; this allows you to divide the jobs among different entities so that if something is going wrong, you might easily know where the problem is. Functions trust each other, so much that one function does not have to know HOW the other

function performs its assignment. One function simply needs to know what the other function needs and supply it.

Once a function has been defined, other functions can use the result of its assignment. Imagine you define two functions A and B.



If Function A needs to use the result of Function B, function A has to use the name of function B. This means Function A needs to "call" Function B:



When calling one function from another function, provide neither the return value nor the body, simply type the name of the function and its list of arguments, if any. For example, to call a function named `Welcome()` from the `main()` function, simply type it, like this:

```
int main(int argc, char* argv[])
{
    Message(); // Calling the Message() function
    return 0;
}
```

The compiler treats the calling of a function depending on where the function is declared with regards to the caller. You can declare a function before calling it. Here is an example:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
void Message()
{
    cout << "This is C++ in its truest form.";
}
//-----
int main(int argc, char* argv[])
{
    Message(); // Calling the Message() function

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
```

If a function is defined after its caller, you should declare it inside of the caller first. Here is an example:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop
```

```
//-----
#pragma argsused
//-----
int main(int argc, char* argv[])
{
    void Message();

    cout << "We will start with the student registration process.\n";
    Message(); // Calling the Message() function

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
void Message()
{
    cout << "Welcome to the Red Oak High School.";
}
//-----
```

To use any of the functions that ships with the compiler, simply call it. As you know already, we have been using the `getchar()` function:

```
int main(int argc, char* argv[])
{
    getchar();
    return 0;
}
```

◆ Calling Functions

1. From what we have learned so far, change the program as follows:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
void Message()
{
    cout << "Welcome to the Red Oak High School.";
}
//-----
int main(int argc, char* argv[])
{
    Message();

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
```

2. To test the program, on the main menu, click Run -> Run.
3. To return to Bcb, press any key.
4. To save the program, on the Standard toolbar, click the Save All button.

void Functions

A function that does not return a value is declared and defined as void. Here is an example:

```
void Introduction()
{
    cout << "This program is used to calculate the areas of some shapes.\n"
        << "The first shape will be a square and the second, a rectangle.\n"
        << "You will be requested to provide the dimensions and the program "
        << "will calculate the areas";
}
```

Any function could be a void type as long as you are not expecting it to return a specific value. A void function with a more specific assignment could be used to calculate and display the area of a square. Here is an example:

```
void SquareArea()
{
    double Side;

    cout << "\nEnter the side of the square: ";
    cin >> Side;

    cout << "\nSquare characteristics:";
    cout << "\nSide = " << Side;
    cout << "\nArea = " << Side * Side;
}
```

When a function is of type void, it cannot be displayed as part of the cout extractor and it cannot be assigned to a variable (since it does not return a value). Therefore, a void function can only be called.

✦ Using void Functions

1. Create a new C++ Console Application based on the Console Wizard.
2. Save the program in a new folder named Function2. Accept the suggested names of the unit and the project.
3. Change the content of the file as follows:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
void FahrenheitToCelsius()
{
    int Fahrenheit, Celsius;

    cout << "Conversion from Fahrenheit to Celsius\n";
    cout << "Type the temperature in Fahrenheit: ";
    cin >> Fahrenheit;
    Celsius = 5 * (Fahrenheit - 32) / 9;
    cout << endl << Fahrenheit << "F = " << Celsius << "C";
}
//-----
int main(int argc, char* argv[])
{
    FahrenheitToCelsius();

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
```

```
}
//-----
```

4. To test the program, on the main menu, press Run -> Run

Returning Values

If you declare a function that is returning anything else than void, the compiler will need to know what value the function returns. The return value must be the same type declared. The value is set with the return keyword.

If a function is declared as a char, make sure it returns a character (only one character). Here is an example:

```
char Answer()
{
    char a;

    cout << "Do you consider yourself a reliable employee (y=Yes/n=No)? ";
    cin >> a;

    return a;
}
```

A good function can also handle a complete assignment and only hand a valid value to other desired functions. Imagine you want to process member's applications at a sports club. You can define a function that would request the first and last names; other functions that need a member's full name would request it from such a function without worrying whether the name is complete. The following function is in charge of requesting both names. It returns a full name that any desired function can use:

```
string GetMemberName()
{
    string FName, LName, FullName;

    cout << "New Member Registration.\n";
    cout << "First Name: ";
    cin >> FName;
    cout << "Last Name: ";
    cin >> LName;

    FullName = FName + " " + LName;
    return FullName;
}
```

✦ Techniques of Returning Values

1. To apply a basic technique of returning a value, change the content of the file as follows:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
int GetFahrenheit()
{
    int Temp;

    cout << "Type the temperature in Fahrenheit: ";
}
```

```

        cin >> Temp;

        return Temp;
    }
    //-----
int main(int argc, char* argv[])
{
    int Celsius, Fahrenheit;

    cout << "This program allows you to convert a temperature "
         << "from Fahrenheit to Celsius\n";
    Fahrenheit = GetFahrenheit();

    Celsius = 5 * (Fahrenheit - 32) / 9;
    cout << endl << Fahrenheit << "F = " << Celsius << "C";

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----

```

2. To test the program, on the main menu, click Run -> Run.
3. After testing the program, press Enter to return to Bcb.
4. For another example, change the program as follows:

```

//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
int GetFahrenheit()
{
    int Temp;

    cout << "Type the temperature in Fahrenheit: ";
    cin >> Temp;

    return Temp;
}
//-----
int GetCelsius()
{
    int Temp;

    cout << "Type the temperature in Celsius: ";
    cin >> Temp;

    return Temp;
}
//-----
int main(int argc, char* argv[])
{
    int Celsius, Fahrenheit;

    cout << "This program allows you to convert a temperature "
         << "from Fahrenheit to Celsius\n";
    Fahrenheit = GetFahrenheit();

    Celsius = 5 * (Fahrenheit - 32) / 9;
    cout << endl << Fahrenheit << "F = " << Celsius << "C";

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}

```

```

    }
    //-----

```

5. To test the program, press F9.

Techniques of Passing Arguments

In order to perform its assignment, a function may need arguments. Any function that wants to use the result of another function must supply the other function's required arguments, if any.

When declaring a function that uses arguments, specify each argument with a data type and a name. Here are examples of declaring functions that take arguments:

```

void SetGender(char a);
double RectangleArea(double L, double W);
char Admission(string Name, double Grade, char g);

```

Passing Arguments by Value

To use a function inside of another function, that is, to call a function from another function, specify the name of the function and its list of arguments (if any) inside of parentheses; only the name of each argument is needed. You can declare a function like this:

```
float GetHours(string FullName);
```

To call such a function from another, you would use:

```
GetHours(FullName);
```

Here is an example:

```

//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
string GetName()
{
    string FirstName, LastName, FN;

    cout << "Employee's First Name: ";
    cin >> FirstName;
    cout << "Employee's Last Name: ";
    cin >> LastName;

    FN = FirstName + " " + LastName;
    return FN;
}
//-----
int main(int argc, char* argv[])
{
    string FullName;
    double Hours;
    double GetHours(string FullName);

    FullName = GetName();
    Hours = GetHours(FullName);

    cout << "\nEmployee's Name: " << FullName;
    cout << "\nWeekly Hours: " << Hours << " hours";
}

```

```

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
double GetHours(string FullName)
{
    double Mon, Tue, Wed, Thu, Fri, TotalHours;

    cout << endl << FullName << "'s Weekly Hours\n";
    cout << "Monday: "; cin >> Mon;
    cout << "Tuesday: "; cin >> Tue;
    cout << "Wednesday: "; cin >> Wed;
    cout << "Thursday: "; cin >> Thu;
    cout << "Friday: "; cin >> Fri;

    TotalHours = Mon + Tue + Wed + Thu + Fri;
    return TotalHours;
}
//-----

```

Here is an example of running the program:

```
Employee's First Name: Frank
Employee's Last Name: Dassault
```

```
Frank Dassault's Weekly Hours
Monday: 8.00
Tuesday: 8.50
Wednesday: 9.00
Thursday: 8.00
Friday: 8.00
```

```
Employee's Name: Frank Dassault
Weekly Hours: 41.5 hours
```

```
Press any key to continue...
```

When declaring a function, the compiler does not require that you supply a name for each argument, it only needs to know what type of argument(s) and how many arguments a function takes. This means the GetHours() function could have been declared as

```
float GetHours(string);
```

Furthermore, the compiler does not care about the name you give an argument when declaring a function. Imagine you want to write a function that would calculate an item's purchase price based on the item's store price added the tax. The tax rate is a percentage value. This means a tax rate set at 7.50% in C++ terms is equal to 0.075 (because 7.50/100 = 0.075). The tax amount collected on a purchase is taken from an item's price; its formula is:

$$\text{Tax Amount} = \text{Item Price} * \frac{\text{TaxRate}}{100}$$

The formula of calculating the final price of an item is:

Final Price = Item Price + Tax Amount

Here is an example:

```

//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

```

```

//-----
#pragma argsused
//-----
int main(int argc, char* argv[])
{
    double ItemPrice, TaxRate;
    double PurchasePrice(double ItemPrice, double TaxRate);

    cout << "Enter the price of the item: ";
    cin >> ItemPrice;
    cout << "Enter the tax rate: ";
    cin >> TaxRate;

    cout << "\nThe final price is: " << PurchasePrice(ItemPrice, TaxRate);

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
double PurchasePrice(double ItemPrice, double TaxRate)
{
    double Price;

    Price = ItemPrice + (ItemPrice * TaxRate / 100);
    return Price;
}
//-----

```

Here is an example of running the program:

```
Enter the price of the item: 128.55
Enter the tax rate: 7.55
```

```
The final price is: 138.256
```

```
Press any key to continue...
```

◆ Passing Arguments by Value

1. To implements an example of passing an argument to a function, change the content of the file as follows:

```

//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop
//-----
#pragma argsused
int GetFahrenheit()
{
    int Temp;

    cout << "Type the temperature in Fahrenheit: ";
    cin >> Temp;

    return Temp;
}
//-----
int main(int argc, char* argv[])
{
    int Celsius, Fahrenheit;
    void FahrenheitToCelsius(int);

```

```


    cout << "This program allows you to convert a temperature "
        << "from one type to another\n";

    Fahrenheit = GetFahrenheit();
    FahrenheitToCelsius(Fahrenheit);

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
void FahrenheitToCelsius(int Fahr)
{
    int Celsius;

    Celsius = 5 * (Fahr - 32) / 9;
    cout << endl << Fahr << "F = " << Celsius << "C";
}
//-----

```

2. To test the program, on the Debug toolbar, click Run .
3. After testing the program, return to Bcb.
4. To pass an argument to a function that also returns an argument, change the following as follows:

```

//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop
//-----

#pragma argsused
int GetFahrenheit()
{
    int Temp;

    cout << "Type the temperature in Fahrenheit: ";
    cin >> Temp;

    return Temp;
}
//-----
int GetCelsius()
{
    int Temp;

    cout << "Type the temperature in Celsius: ";
    cin >> Temp;

    return Temp;
}
//-----
int main(int argc, char* argv[])
{
    int Celsius, Fahrenheit;
    void FahrenheitToCelsius(int);
    int CelsiusToFahrenheit(int);

    cout << "This program allows you to convert a temperature "
        << "from one type to another\n";
    Fahrenheit = GetFahrenheit();
    FahrenheitToCelsius(Fahrenheit);

    int Cels, Fahren;

    Cels = GetCelsius();
}

```

```

    Fahren = CelsiusToFahrenheit(Cels);


    cout << endl << Cels << "C = " << Fahren << "F";

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
void FahrenheitToCelsius(int Fahr)
{
    int Celsius;

    Celsius = 5 * (Fahr - 32) / 9;
    cout << endl << Fahr << "F = " << Celsius << "C\n\n";
}
//-----
int CelsiusToFahrenheit(int Cel)
{
    int Fahr;

    Fahr = ((9 * Cel) / 5) + 32;
    return Fahr;
}
//-----

```

5. To test the program, on the Debug toolbar, click Run .
6. After testing the program, return to Bcb.
7. To pass two arguments to a function, change the program as follows:

```

//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop
//-----

#pragma argsused
int GetFahrenheit()
{
    int Temp;

    cout << "Type the temperature in Fahrenheit: ";
    cin >> Temp;

    return Temp;
}
//-----
int GetCelsius()
{
    int Temp;

    cout << "Type the temperature in Celsius: ";
    cin >> Temp;

    return Temp;
}
//-----
int main(int argc, char* argv[])
{
    int Celsius, Fahrenheit;
    void FahrenheitToCelsius(int, int);
    void CelsiusToFahrenheit(int, int);

    cout << "This program allows you to convert a temperature "
        << "from one type to another\n";

    Fahrenheit = GetFahrenheit();
}

```

```

    FahrenheitToCelsius(Fahrenheit, Celsius);

    int Cels, Fahr;

    Cels = GetCelsius();
    CelsiusToFahrenheit(Cels, Fahr);

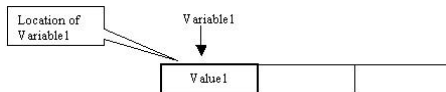
    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
void FahrenheitToCelsius(int Fahr, int Cels)
{
    Cels = 5 * (Fahr - 32) / 9;
    cout << endl << Fahr << "F = " << Cels << "C\n\n";
}
//-----
void CelsiusToFahrenheit(int Cel, int Fahr)
{
    Fahr = ((9 * Cel) / 5) + 32;
    cout << endl << Cel << "C = " << Fahr << "F";
}
//-----

```

8. Test the program. Return to Bcb.

Passing Arguments by Reference

When you declare a variable in a program, the compiler reserves an amount of space for that variable. If you need to use that variable somewhere in your program, you call it and make use of its value. There are two major issues related to a variable: its value and its location in the memory:



The location of a variable in memory is referred to as its address.

If you supply the argument using its name, the compiler only makes a copy of the argument's value and gives it to the calling function. Although the calling function receives the argument's value and can use in any way, it cannot (permanently) alter it. C++ allows a calling function to modify the value of a passed argument if you find it necessary. If you want the calling function to modify the value of a supplied argument and return the modified value, you should pass the argument using its reference.

To pass an argument as a reference, when declaring the function, precede the argument name with an ampersand "&". You can pass one or more arguments as reference in the program or pass all arguments as reference. The decision as to which argument(s) should be passed by value or by reference is based on whether or not you want the called function to modify the argument and permanently change its value.

Here are examples of passing some arguments by reference:

```

void Area(double &Side); // The argument is passed by reference
bool Decision(char &Answer, int Age); // One argument is passed by reference
// All arguments are passed by reference
float Purchase(float &DiscountPrice, float &NewDiscount, char &Commission);

```

You add the ampersand when declaring a function and/or when defining it. When calling the function, supply only the name of the referenced argument(s). The

above would be called with:

```

Area(Side);
Decision(Answer, Age);
Purchase(DiscountPrice, NewDiscount, Commission);

```

You will usually need to know what happens to the value passed to a calling function because the rest of the program may depend on it.

Imagine that you write a function that calculates employees weekly salary provided the total weekly hours and hourly rate. To illustrate our point, we will see how or whether one function can modify a salary of an employee who claims to have worked more than the program displays. The starting regular program would be as follows:

```

//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
//-----
int main(int argc, char* argv[])
{
    float Hours, Rate, Wage;
    void Earnings(float h, float r);

    cout << "Enter the total Weekly hours: ";
    cin >> Hours;
    cout << "Enter the employee's hourly rate: ";
    cin >> Rate;

    cout << "\n\nIn the main() function,";
    cout << "\n\tWeekly Hours = " << Hours;
    cout << "\n\tSalary = " << Rate;
    cout << "\n\tWeekly Salary: " << Hours * Rate;

    cout << "\nCalling the Earnings() function";
    Earnings(Hours, Rate);
    cout << "\n\nAfter calling the Earnings() function, "
        << "in the main() function,";

    cout << "\n\tWeekly Hours = " << Hours;
    cout << "\n\tSalary = " << Rate;
    cout << "\n\tWeekly Salary: " << Hours * Rate;

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
void Earnings(float ThisWeek, float Salary)
{
    cout << "\n\nIn the Earnings() function,";
    cout << "\n\tWeekly Hours = " << ThisWeek;
    cout << "\n\tSalary = " << Salary;
    cout << "\n\tWeekly Salary= " << ThisWeek * Salary;
}
//-----

```

If you test the program by typing 32 for the weekly hours and 6.45 for the salary, you would notice the weekly values are the same.

Imagine that the employee claims to have worked 42 hours instead of the passed weekly hours. You could create the following function to find out.


```
//-----
void Earnings(float ThisWeek, float Salary)
{
    ThisWeek = 42;
    cout << "\n\nIn the Earnings() function,";
    cout << "\n\tWeekly Hours = " << ThisWeek;
    cout << "\n\tSalary = " << Salary;
    cout << "\n\tWeekly Salary= " << ThisWeek * Salary;
}
//-----
```

If you test the program with a weekly hours value of 35.50 and a salary of 8.50, you would notice that the weekly salary is different inside of the Earnings() function but is kept the same in main(), before and after the Earnings() function:

```
Enter the total Weekly hours: 35.50
Enter the employee's hourly rate: 8.50
```

```
In the main() function,
    Weekly Hours = 35.5
    Salary = 8.5
    Weekly Salary: 301.75
Calling the Earnings() function
```

```
In the Earnings() function,
    Weekly Hours = 42
    Salary = 8.5
    Weekly Salary= 357
```

```
After calling the Earnings() function, in the main() function,
    Weekly Hours = 35.5
    Salary = 8.5
    Weekly Salary: 301.75
```

```
Press any key to continue...
```

As an example of passing an argument by reference, you could modify the declaration of the Earnings() function inside of the main() function as follows:

```
void Earnings(float &h, float r);
```

If you want a calling function to modify the value of an argument, you should supply its reference and not its value. You could change the function as follows:

```
//-----
void Earnings(float &ThisWeek, float Salary)
{
    ThisWeek = 42;
    cout << "\n\nIn the Earnings() function,";
    cout << "\n\tWeekly Hours = " << ThisWeek;
    cout << "\n\tSalary = " << Salary;
    cout << "\n\tWeekly Salary= " << ThisWeek * Salary;
}
//-----
```

❖ Passing Arguments by Reference

1. To apply the passing of an argument by reference, change the file as follows:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop
//-----
```

```
#pragma argsused
void GetFahrenheit(int& Temp)
{
    cout << "Type the temperature in Fahrenheit: ";
    cin >> Temp;
}
//-----
void GetCelsius(int& Temp)
{
    cout << "Type the temperature in Celsius: ";
    cin >> Temp;
}
//-----
int main(int argc, char* argv[])
{
    int Celsius, Fahrenheit;
    void FahrenheitToCelsius(int, int);
    void CelsiusToFahrenheit(int, int);

    cout << "This program allows you to convert a temperature "
        << "from one type to another\n";
    GetFahrenheit(Fahrenheit);
    FahrenheitToCelsius(Fahrenheit, Celsius);

    int Cels, Fahren; GetCelsius(Cels);
    CelsiusToFahrenheit(Cels, Fahren);

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
void FahrenheitToCelsius(int Fahr, int Cels)
{
    Cels = 5 * (Fahr - 32) / 9;

    cout << endl << Fahr << "F = " << Cels << "C\n\n";
}
//-----
void CelsiusToFahrenheit(int Cel, int Fahr)
{
    Fahr = ((9 * Cel) / 5) + 32;
    cout << endl << Cel << "C = " << Fahr << "F";
}
//-----
```

2. To test the program, on the main menu, click Run ^a Run.
3. After testing the program, return to Bcb.

Default Arguments

We have seen that some functions take one or more arguments. Whenever a function takes an argument, that argument is required. If the calling function does not provide the (required) argument, the compiler would throw an error.

Imagine you write a function that will be used to calculate the final price of an item after discount. The function would need the discount rate in order to perform the calculation. Such a function could look like this:

```
double CalculateNetPrice(double DiscountRate)
{
    double OrigPrice;

    cout << "Please enter the original price: ";
    cin >> OrigPrice;

    return OrigPrice - (OrigPrice * DiscountRate / 100);
}
```

```
}
```

Since this function expects an argument, if you do not supply it, the following program would not compile:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
double CalculateNetPrice(double DiscountRate)
{
    double OrigPrice;

    cout << "Please enter the original price: ";
    cin >> OrigPrice;

    return OrigPrice - (OrigPrice * DiscountRate / 100);
}
//-----
int main(int argc, char* argv[])
{
    double FinalPrice;
    double Discount = 15; // That is 25% = 25

    FinalPrice = CalculateNetPrice(Discount);

    cout << "\nAfter applying the discount";
    cout << "\nFinal Price = " << FinalPrice << "\n";

    cout << "\nPress any key to continue...";
    getch();
    return 0;
}
//-----
```

Here is an example of running the program:

Please enter the original price: 255.95

After applying the discount
Final Price = 217.558

Press any key to continue...

Most of the time, a function such as ours would use the same discount rate over and over again. Therefore, instead of supplying an argument all the time, C++ allows you to define an argument whose value would be used whenever the function is not provided with the argument.

To give a default value to an argument, when declaring the function, type the name of the argument followed by the = sign, followed by the default value. The CalculateNetPrice() function, with a default value, could be defined as:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
double CalculateNetPrice(double DiscountRate = 25)
```

```
{
    double OrigPrice;

    cout << "Please enter the original price: ";
    cin >> OrigPrice;

    return OrigPrice - (OrigPrice * DiscountRate / 100);
}
//-----
int main(int argc, char* argv[])
{
    double FinalPrice;

    FinalPrice = CalculateNetPrice();

    cout << "\nAfter applying the discount";
    cout << "\nFinal Price = " << FinalPrice << "\n";

    cout << "\nPress any key to continue...";
    getch();
    return 0;
}
//-----
```

Here is an example of running the program:

Please enter the original price: 120.15

After applying the discount
Final Price = 90.1125

Press any key to continue...

If a function takes more than one argument, you can provide a default argument for each and select which ones would have default values. If you want all arguments to have default values, when defining the function, type each name followed by = followed by the desired value. Here is an example:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
double CalculateNetPrice(double Tax = 5.75, double Discount = 25,
                        double OrigPrice = 245.55)
{
    double DiscountValue = OrigPrice * Discount / 100;
    double TaxValue = Tax / 100;
    double NetPrice = OrigPrice - DiscountValue + TaxValue;

    cout << "Original Price: $" << OrigPrice << endl;
    cout << "Discount Rate: " << Discount << "%" << endl;
    cout << "Tax Amount: $" << Tax << endl;

    return NetPrice;
}
//-----
int main(int argc, char* argv[])
{
    double FinalPrice;

    FinalPrice = CalculateNetPrice();

    cout << "\nAfter applying the discount";
    cout << "\nFinal Price = " << FinalPrice << "\n";
```

```

    cout << "\nPress any key to continue...";
    getch();
    return 0;
}
//-----

```

Here is the result produced:

```

Original Price: $245.55
Discount Rate: 25%
Tax Amount: $5.75

After applying the discount
Final Price = 184.22

Press any key to continue...

```

If a function takes more than one argument and you would like to provide default values for those parameters, the order of appearance of the arguments is very important.

1/ If a function takes two arguments, you can declare it with default values. We already know how to do that. If you want to provide a default value for only one of the arguments, the argument that would have a default value must be the second in the list. Here is an example:

```
double CalculatePrice(double Tax, double Discount = 25);
```

When calling such a function, if you supply only one argument, the compiler would assign its value to the first parameter in the list and ignore assigning a value to the second:

```

//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
double CalculateNetPrice(double Tax, double Discount = 25)
{
    double OrigPrice, DiscountValue, TaxValue, NetPrice;

    cout << "Enter the original price of the item: ";
    cin >> OrigPrice;

    DiscountValue = OrigPrice * Discount / 100;
    TaxValue = Tax / 100;
    NetPrice = OrigPrice - DiscountValue + TaxValue;

    return NetPrice;
}
//-----
int main(int argc, char* argv[])
{
    double FinalPrice;
    double TaxRate = 5.50; // = 5.50%

    FinalPrice = CalculateNetPrice(TaxRate);

    cout << "\nAfter applying the discount";
    cout << "\nFinal Price = " << FinalPrice << "\n";

    cout << "\nPress any key to continue...";
    getch();
}

```

```

    return 0;
}
//-----

```

Here is an example of running the program:

```

Enter the original price of the item: 250.50

After applying the discount
Final Price = 187.93

Press any key to continue...

```

If you define the function and assign a default value to the first argument, if you provide only one argument when calling the function, you would receive an error.

2/ If the function receives more than two arguments and you would like only some of those arguments to have default values, the arguments that would have default values must be at the end of the list. Regardless of how many arguments would or would not have default values, start the list of arguments without those that would not use default values.

◆ Using Default Arguments

1. Create a new C++ Console Application based on the Console Wizard
2. Save it in a new folder named Default
3. To declare a function with one default argument, change the file as follows:

```

//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
//-----
int main(int argc, char* argv[])
{
    float Length, Width, Height;
    float BoxArea(float l = 25.50, float w = 16.25, float h = 8.95);
    cout << "With default values, Cube Area = " << BoxArea();

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
float BoxArea(float Len, float Wd, float Ht)
{
    float Area;

    Area = Len * Wd * Ht;
    return Area;
}
//-----

```

4. To test the program, press F9

Function Overloading

A C++ program involves a great deal of names that represent variables and functions of various kinds. The compiler does not allow two entities to share a name; for example, two variables must not have the name in the same function.

Although two functions should have unique names in the same program, C++ allows you to use the same name for different functions of the same program following certain rules. The ability to have various functions with the same name in the same program is called function overloading.

The most important rule about function overloading is to make sure that each one of these functions has a different number or different types of arguments. For example, you can create a function called Area() but define it to calculate the areas of different shapes:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
// Area of a square
float Area(float Side)
{
    return (Side * Side);
}
//-----
// Area of a rectangle
float Area(float Length, float Width)
{
    return (Length * Width);
}
//-----
int main(int argc, char* argv[])
{
    float s, l, w;

    s = 15.25;
    l = 28.12;
    w = 10.35;

    cout << "The rea of the square is " << Area(s);
    cout << "\nThe area of the rectangle is " << Area(l, w);

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
```

Here is the result of running the program:

```
The rea of the square is 232.562
The area of the rectangle is 291.042

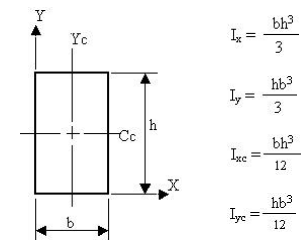
Press any key to continue...
```

❖ Overloading Functions

The moment of inertia is the ability of of a beam to resist bending. It is calculated with regard to the cross section of the beam. Because it depends on the type of section of the beam, its calculation also depends on the type of section of the beam. In this exercise, we will review different formulas used to calculate the moment of inertia. Since this exercise is for demonstration purposes, you do not need to be a Science Engineering major to understand it.

1. Create a new Console Application using the Console Wizard.
2. Here is the formulas to calculate the moment of inertia of a rectangle depending

on the axis considered:



To calculate the moment of inertia with regard to the X axis, change the file as follows:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----

#pragma argsused
//-----
// Rectangle
double MomentOfInertia(double b, double h)
{
    return b * h * h * h / 3;
}
//-----
int main(int argc, char* argv[])
{
    double Base, Height;

    cout << "Enter the dimensions of the Rectangle\n";
    cout << "Base: ";
    cin >> Base;
    cout << "Height: ";
    cin >> Height;

    cout << "\nMoment of inertia with regard to the X axis: ";
    cout << "I = " << MomentOfInertia(Base, Height) << "mm";

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
```

3. To test the function, on the main menu, click Run -> Run. Here is an example:

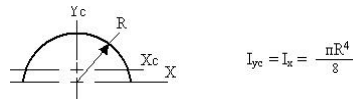
```
Enter the dimensions of the Rectangle
Base: 12.74
Height: 7.28

Moment of inertia with regard to the X axis: I = 1638.48mm

Press any key to continue...
```

4. After testing the function, return to Bcb.
5. Here are the formulas to calculate the moment of inertia for a semi-circle:

$$I_{xc} = 0.110R^4$$



A circle and thus a semi-circle requires only a radius. Since the other version of the MomentOfInertia() function requires two argument, we can overload it by providing only one argument, the radius. To calculate the moment of inertia with regard to the X or base axis, overload the MomentOfInertia() function as follows:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----
#pragma argsused
//-----
// Rectangle
double MomentOfInertia(double b, double h)
{
    return b * h * h * h / 3;
}
//-----
// Semi-Circle
double MomentOfInertia(double R)
{
    const double PI = 3.14159;

    return R * R * R * R * PI / 8;
}
//-----
int main(int argc, char* argv[])
{
    double Base, Height, Radius;

    cout << "Enter the dimensions of the Rectangle\n";
    cout << "Base: ";
    cin >> Base;
    cout << "Height: ";
    cin >> Height;

    cout << "Moment of inertia with regard to the X axis: ";
    cout << "I = " << MomentOfInertia(Base, Height) << "mm";

    cout << "\n\nEnter the radius: ";
    cin >> Radius;
    cout << "Moment of inertia of a semi-circle with regard to the X axis: ";
    cout << "I = " << MomentOfInertia(Radius) << "mm\n";

    cout << "\n\nPress any key to continue...";
    getch();
    return 0;
}
//-----
```

6. Test the program. Here is an example:

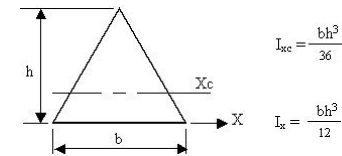
```
Enter the dimensions of the Rectangle
Base: 28.55
Height: 18.75
Moment of inertia with regard to the X axis: I = 62731.9mm

Enter the radius: 22.52
Moment of inertia of a semi-circle with regard to the X axis: I = 101003mm
```

Press any key to continue...

7. After testing the program, return to Bcb

8. Here are the formulas to calculate the moment of inertia of a triangle:



As you can see, the rectangle and the triangle are using the same dimension types. This means, we can provide only the same kinds of arguments, the base and the height, to calculate the moment of inertia. In order to overload the MomentOfInertia() function, we will add an argument that will never be used; this argument will serve only as a "witness" to set the difference between both versions of the function. This "witness" argument can be anything: an integer, a character, a string, a float, etc. For our example, we will make it a simple integer. To use the version applied to the triangle, we will provide this argument to overload the MomentOfInertia() function. When called with only two arguments, the rectangle version will apply. To calculate the moment of inertia with regard to the X axis, overload the MomentOfInertia function as follows:

```
//-----
#include <iostream>
#include <conio>
using namespace std;
#pragma hdrstop

//-----
#pragma argsused
//-----
// Rectangle
double MomentOfInertia(double b, double h)
{
    return b * h * h * h / 3;
}
//-----
// Semi-Circle
double MomentOfInertia(double R)
{
    const double PI = 3.14159;

    return R * R * R * R * PI / 8;
}
//-----
// Triangle
double MomentOfInertia(double b, double h, int)
{
    return b * h * h * h / 12;
}
//-----
int main(int argc, char* argv[])
{
    double Base = 7.74,
           Height = 14.38,
           Radius = 12.42;

    cout << "Rectangle\n"
          << "Moment of inertia with regard to the X axis: ";
}
```

```
cout << "I = " << MomentOfInertia(Base, Height) << "mm\n\n";

cout << "Semi-Circle\n"
    << "Moment of inertia with regard to the X axis: ";
cout << "I = " << MomentOfInertia(Radius) << "mm\n\n";

cout << "Enter the dimensions of the triangle\n";
cout << "Base: ";
cin >> Base;
cout << "Height: ";
cin >> Height;
cout << "\nTriangle\n"
    << "Moment of inertia with regard to the X axis: ";
cout << "I = " << MomentOfInertia(Base, Height, 1) << "mm\n";

cout << "\n\nPress any key to continue...";
getch();
return 0;
}
//-----
```

9. Test the program. Here is an example:

```
Rectangle
Moment of inertia with regard to the X axis: I = 7671.78mm

Semi-Circle
Moment of inertia with regard to the X axis: I = 9344.28mm

Enter the dimensions of the triangle
Base: 18.24
Height: 10.78

Triangle
Moment of inertia with regard to the X axis: I = 1904.14mm

Press any key to continue...
```

10. After testing the program, return to Bcb