Exploring Functions

Constant Arguments

When a function receives an argument, it performs one of two actions with regards to the value of the argument; it might modify the value itself or only use the argument to modify another argument or another of its own variables. If you know that the function is not supposed to alter the value of an argument, you should let the compiler know. This is a safeguard that serves at least two purposes. First, the compiler will make sure that the argument supplied stays intact; if the function tries to modify the argument, the compiler would throw an error, letting you know that an undesired operation took place. Second, this speeds up execution.

To let the compiler know that the value of an argument must stay constant, use the const keyword before the data type of the argument. For example, if you declare a function like void Area(const string Side), the Area() is not supposed to modify the value of the Side argument. Consider a function that is supposed to calculate and return the perimeter of a rectangle if it receives the length and the width from another function, namely main(). Here is a program that would satisfy the operation (notice the Perimeter() function that takes two arguments):

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
//---------------------------------------------------------------------------
float Perimeter(float l, float w)
{
    double p;

    p = 2 * (l * w);
    return p;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    float Length, Width;

    cout << "Rectangle dimensions.\n";
    cout << "Enter the length: ";
    cin >> Length;
    cout << "Enter the width: ";
    cin >> Width;
    cout << "\nThe perimeter of the rectangle is: "
         << Perimeter(Length, Width);

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

This would produce:

```
Rectangle dimensions.
Enter the length: 35.55
Enter the width: 28.75
```



```
The perimeter of the rectangle is: 2044.12

Press any key to continue...
```

As you can see, the Perimeter() function does not change the values of the length or the width. To reinforce the purpose of the assignment, you should make this clear to the compiler. To make the length and the width arguments constant, you would change the declaration of the Perimeter() function as follows:

```
float Perimeter(const float l, const float w);
```

You can make just one or more arguments constants, and there is no order on which arguments can be made constant.

Using Constant Arguments

1. Start Borland C++ Builder and create a new C++ Console Application using the Console Wizard.
2. To apply the constantness of arguments passed to functions, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

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

    return R * R * R * R * PI/ 8;
}
//---------------------------------------------------------------------------
// Triangle
double MomentOfInertia(const double b, const 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 << "\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...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

3. To test the program, on the main menu, click Run -> Run:

```
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


Triangle
Moment of inertia with regard to the X axis: I = 1917.95mm


Press any key to continue...
```

4. To use a mix of functions, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

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

    return R * R * R * R * PI/ 8;
}
//---------------------------------------------------------------------------
// Triangle
double MomentOfInertia(const double b, const double h, int)
{
    return b * h * h * h / 12;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    double Length, Height, Radius;
    double GetBase();
    double GetHeight();
    double GetRadius();

    cout << "Enter the dimensions of the rectangle\n";
    Length = GetBase();
    Height = GetHeight();
    cout << "Rectangle\n"
         << "Moment of inertia with regard to the X axis: ";
    cout << "I = " << MomentOfInertia(Length, Height) << "mm\n\n";

    cout << "Enter the radius of the semi-circle\n";
    Radius = GetRadius();
    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";
    Length = GetBase();
    Height = GetHeight();

    cout << "\nTriangle\n"
         << "Moment of inertia with regard to the X axis: ";
    cout << "I = " << MomentOfInertia(Length, Height, 1) << "mm\n";

    cout << "\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
double GetBase()
{
    double B;

    cout << "Enter Base: ";
    cin >> B;

    return B;
}
//---------------------------------------------------------------------------
double GetHeight()
{
    double H;

    cout << "Enter Height: ";
    cin >> H;
    return H;
}
//---------------------------------------------------------------------------
double GetRadius()
{
    double R;

    cout << "Enter Radius: ";
    cin >> R;
    return R;
}
//---------------------------------------------------------------------------
```

5. Test the program. Here is an example:

```
Enter the dimensions of the rectangle
Enter Base: 18.25
Enter Height: 14.15
Rectangle
Moment of inertia with regard to the X axis: I = 17235mm

Enter the radius of the semi-circle
Enter Radius: 15.55
Semi-Circle
Moment of inertia with regard to the X axis: I = 22960.5mm

Enter the dimensions of the triangle
Enter Base: 16.35
Enter Height: 12.75

Triangle
Moment of inertia with regard to the X axis: I = 2824.02mm

Press any key to continue...
```

6. After examining the program, return to Bcb.

Passing Arguments by Constant Reference

We have seen that passing an argument as a reference allows the compiler to retrieve the real value of the argument at its location rather than sending a request for a copy of the variable. This speeds up the execution of the program. Also, when passing an argument as a constant, the compiler will make sure that the value of the passed argument is not modified. What would happen if you combine both techniques?

If you pass an argument as reference, the compiler would access the argument from its location. The called function can modify the value of the argument. The advantage is that the code execution is faster because the argument gives access to its address. The disadvantage could be that if the calling function modifies the value of the argument, when the function exits, the value of the argument would have (permanently) changed and the original value would be lost (actually, this can be an advantage as we have learned in the passed). If you do not want the value of the passed argument to be modified, you should pass the argument as a constant reference. When doing this, the compiler would access the argument at its location (or address) but it would make sure that the value of the argument stays intact.

To pass an argument as a constant reference, when declaring the function and when implementing it, type the const keyword, followed by the argument data type, followed by the ampersand operator, followed by a name for the argument. When declaring the function, the name of the argument is optional. Here is a function that receives an argument as a constant reference:

```
//---------------------------------------------------------------------------
double CalculateNetPrice(const double& Tax)
{
    double Original;
    const double Discount = 25;

    Original = GetOriginalPrice();
    double DiscountValue = Original * Discount / 100;
    double TaxValue = Tax / 100;
    double NetPrice = Original - DiscountValue + TaxValue;

    return NetPrice;
}
//---------------------------------------------------------------------------
```

You can mix arguments passed by value, those passed as reference, those passed by constant, and those passed by constant references. You will decide, based on your intentions, to apply whatever technique suits your scenario.

The following program illustrates the use of various techniques of passing arguments:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop
//---------------------------------------------------------------------------
#pragma argsused
//---------------------------------------------------------------------------
// Passing an argument by reference
void GetOriginalPrice(double& OriginalPrice)
{
    cout << "Enter the original price of the item: $";
    cin >> OriginalPrice;
}
```



```
//---------------------------------------------------------------------------
// Passing an argument as a constant reference
// Passing arguments by value
double CalculateNetPrice(const double& Original, double Tax, double Discount)
{
    Discount = Original * Discount / 100;
    Tax = Tax / 100;
    double NetPrice = Original - Discount + Tax;

    return NetPrice;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    double TaxRate = 5.50; // = 5.50%
    const double Discount = 25;
    double Price;
    double Original;
    void Receipt(const double& Orig, const double& Taxation,
    const double& Dis, const double& Final);

    GetOriginalPrice(Original);
    Price = CalculateNetPrice(Original, TaxRate, Discount);
    Receipt(Original, TaxRate, Discount, Price);

    cout << "\n\nPress any key to continue...";
    getchar();
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
void Receipt(const double& Original, const double& Tax,
             const double& Discount, const double& FinalPrice)
{
    cout << "\nReceipt";
    cout << "\nOriginal Price: $" << Original;
    cout << "\nTax Rate: " << Tax << "%";
    cout << "\nDiscount Rate: " << Discount << "%";
    cout << "\nFinal Price: $" << FinalPrice;
}
//---------------------------------------------------------------------------
```

Passing Arguments by Constant References

1. To illustrate the passing of arguments by reference and by constant references, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

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

    return R * R * R * R * PI/ 8;
}
//---------------------------------------------------------------------------
```

```
// Triangle
double MomentOfInertia(const double& b, const double& h, const int&)
{
    return b * h * h * h / 12;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    double Length, Height, Radius;
    void GetBaseAndHeight(double&, double&);
    void GetRadius(double&);

    cout << "Enter the dimensions of the rectangle\n";
    GetBaseAndHeight(Length, Height);
    cout << "Rectangle\n"
         << "Moment of inertia with regard to the X axis: ";
    cout << "I = " << MomentOfInertia(Length, Height) << "mm\n\n";

    cout << "Enter the radius of the semi-circle\n";
    GetRadius(Radius);
    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";
    GetBaseAndHeight(Length, Height);

    cout << "\nTriangle\n"
         << "Moment of inertia with regard to the X axis: ";
    cout << "I = " << MomentOfInertia(Length, Height, 1) << "mm\n";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
// Passing arguments by reference
void GetBaseAndHeight(double& B, double& H)
{
    cout << "Enter Base:   ";
    cin >> B;
    cout << "Enter Height: ";
    cin >> H;
}
//---------------------------------------------------------------------------
void GetRadius(double& R)
{
    cout << "Enter Radius: ";
    cin >> R;
}
//---------------------------------------------------------------------------
```

2. To test the program, on the Debug toolbar, click the Run button. Here is an example:

```
Enter the dimensions of the rectangle
Enter Base:   18.85
Enter Height: 15.55
Rectangle
Moment of inertia with regard to the X axis: I = 23625.5mm

Enter the radius of the semi-circle
Enter Radius: 14.25
Semi-Circle
Moment of inertia with regard to the X axis: I = 16192.7mm

Enter the dimensions of the triangle
Enter Base:   8.95
Enter Height: 11.25
```



```
Triangle
Moment of inertia with regard to the X axis: I = 1061.94mm


Press any key to continue...
```

3. After testing the program, return to Bcb.
4. To further mix the passing of arguments, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
//---------------------------------------------------------------------------
// Rectangle
// This function receives one argument by reference and two arguments
// by constant references
void MomentOfInertia(double& Moment,
                     const double& b, const double& h)
{
    Moment = b * h * h * h / 3;
}
//---------------------------------------------------------------------------
// Semi-Circle
// This function receives one argument by reference and one by
// constant reference
void MomentOfInertia(double& Moment, const double& R)
{
    const double PI = 3.14159;

    Moment = R * R * R * R * PI/ 8;
}
//---------------------------------------------------------------------------
// Triangle
// This function receives one argument by reference, two arguments by
// constant references and one argument by value
void MomentOfInertia(double& Moment,
                      const double& b, const double& h, const int&)
{
    Moment = b * h * h * h / 12;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    double Length, Height, Radius, MRectangle, MSemiCircle, MTriangle;
    void GetBaseAndHeight(double&, double&);
    void GetRadius(double&);

    cout << "Enter the dimensions of the rectangle\n";
    GetBaseAndHeight(Length, Height);
    MomentOfInertia(MRectangle, Length, Height);
    cout << "Rectangle\n"
         << "Moment of inertia with regard to the X axis: ";
    cout << "I = " << MRectangle << "mm\n\n";
    cout << "Enter the radius of the semi-circle\n";

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

    GetBaseAndHeight(Length, Height);
    MomentOfInertia(MRectangle, Length, Height, 1);
    cout << "\nTriangle\n"
         << "Moment of inertia with regard to the X axis: ";
```

```
    cout << "I = " << MRectangle << "mm\n";

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
// Passing arguments by reference
void GetBaseAndHeight(double& B, double& H)
{
    cout << "Enter Base:   ";
    cin >> B;
    cout << "Enter Height: ";
    cin >> H;
}
//---------------------------------------------------------------------------
void GetRadius(double& R)
{
    cout << "Enter Radius: ";
    cin >> R;
}
//---------------------------------------------------------------------------
```

5. Test the program. Here is an example:

```
Enter the dimensions of the rectangle
Enter Base:   12.85
Enter Height: 8.85
Rectangle
Moment of inertia with regard to the X axis: I = 2969.01mm

Enter the radius of the semi-circle
Enter Radius: 5.55
Semi-Circle
Moment of inertia with regard to the X axis: I = 372.59mm

Enter the dimensions of the triangle
Enter Base:   10.75
Enter Height: 6.75

Triangle
Moment of inertia with regard to the X axis: I = 275.511mm


Press any key to continue...
```

6. Return to Bcb.

Passing Arguments to Registers

All the variables that we have used so far were declared in, and passed to, the random memory (RAM). Once a variable is declared and "put" in the memory, whenever it is involved in a calculation or assignment, the microprocessor sends a request to the memory to retrieve the value of the variable.

The Central Processing Unit (CPU), also called the microprocessor, has its own memory. The microprocessor is made of memory cells called registers. Unlike the memory in the RAM, the access of the memory in the microprocessor is more precise; so precise that the registers are referred to by using their names. Some of the most commonly used registers (also called general purpose registers) are called EAX, EBX, ECX, EDX, ESI, etc. These registers are mostly used in the Assembly Language for low-level programming. Using registers allows the programmer to write assignments directly destined for the microprocessor. The assignments and operations in the Assembly language are called instructions. When instructions are used by registers, the processing of the program is fast because the microprocessor does not have to retrieve the values of the variables in the RAM; these values, since existing in the registers, are readily available.

C++ Builder (and most popular compilers) allow you to include Assembly Language code in your program. Using this feature, you can write a section or sections of Assembly language. A section that has Assembly code starts with __asm followed by some other techniques. When the compiler encounters this keyword, it knows that the subsequent code would be in Assembly language and it would treat it accordingly. For example, instead of performing a calculation in the RAM, the following program will assign values to two integer variables, namely Number1 and Number2, then it calculate their sum of those two numbers and stores the result in another variable called Result. After the calculation the Assembly section sends the result back to the C++ compiler to display the variables and their values:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[])
{
    int Number1, Number2, Result;

    __asm
    {
        MOV Number1, 248 // Initialize Number1
        MOV Number2, 405 // Initialize Number2
        MOV EAX, Number1 // Put the value of Number1 in the EAX register
        ADD EAX, Number2 // Add the value of Number2 to the content of EAX
        MOV Result, EAX  // Put the content of EAX into Result
    }                    // That's it

    cout << "Number1 = " << Number1 << endl;
    cout << "Number2 = " << Number2 << endl;
    cout << "\nAfter adding Number1 to Number2," << endl;
    cout << "Result  = " << Result << endl;

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

This would produce:

```
Number1 = 248
Number2 = 405

After adding Number1 to Number2,
Result  = 653

Press any key to continue...
```

C++ Builder ships with a version of Assembly language so that if you are interested in adding low-level code, you do not have to purchase an assembler. In fact, C++ installs TASM (known as Turbo Assembler), the award wining Assembler from Borland. This means that, if you want to learn Assembly, you don't have to purchase it anymore (unfortunately, it is not documented). Alternatively, the C++ Builder compiler has its own Assembler known as Inline Assembly. This allows you to embed Assembly code in your programs.

The ability for C++ Builder to recognize Assembly code allows you to pass arguments to registers. For example, you can pass arguments to the EAX, EBX, ECX, or EDX, etc register to speed the compilation process. Fortunately, you do not need to learn Assembly language (although you are encouraged to do so) to speed your code in C++ Builder. As an alternative, you can use the __fastcall keyword

whenever you would have passed arguments in registers.

The syntax for using the __fastcall keyword is:

ReturnType __fastcall FunctionName(Arguments);

Whenever you decide to use __fastcall, use it both when declaring and when defining the function. As an introductory example of using __fastcall, the following program uses two functions. The first function, GetFullName() requests an employee's first and last names, then it returns the full name. Since this function is defined before being used, it was not declared in the main() function. The second function, because defined after main() (I did this on purpose), is declared in main() prior to using it. Both functions use the __fastcall keyword. Notice that both functions have their arguments also passed by reference. Here is the complete program:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop
//---------------------------------------------------------------------------
#pragma argsused
//---------------------------------------------------------------------------
string __fastcall GetFullName(string fn, string ln)
{
    string FN;

    cout << "First Name: ";
    cin >> fn;
    cout << "Last Name: ";
    cin >> ln; FN = fn + " " + ln;

    return FN;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    string FirstName, LastName, FullName;
    double Hours, HourlySalary, WeeklySalary;
    void __fastcall GetHours(double& x, double& y);

    cout << "Enter information about the employee\n";
    FullName = GetFullName(FirstName, LastName);
    GetHours(Hours, HourlySalary);
    WeeklySalary = Hours * HourlySalary;

    cout << "\nEmployee's Records";
    cout << "\nFull Name: " << FullName;
    cout << "\nWeekly Hours: " << Hours;
    cout << "\nHourly Salary: $" << HourlySalary;
    cout << "\nWeekly Wages: $" << WeeklySalary;

    cout << "\n\nPress any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
void __fastcall GetHours(double& h, double& s)
{
    cout << "Total hours for the week: ";
    cin >> h;
    cout << "Hourly Salary: $";
    cin >> s;
```



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

Here is an example of running the program:

```
Enter information about the employee
First Name: Henry
Last Name: Ndjakou
Total hours for the week: 35.50
Hourly Salary: $12.55

Employee's Records
Full Name: Henry Ndjakou
Weekly Hours: 35.5
Hourly Salary: $12.55
Weekly Wages: $445.525

Press any key to continue...
```

Using __fastcall

1. To use __fastcall, change the program as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
//---------------------------------------------------------------------------
// Rectangle
// This function receives one argument by reference and two arguments
// by constant references
void __fastcall MomentOfInertia(double& Moment,
                    const double& b, const double& h)
{
    Moment = b * h * h * h / 3;
}
//---------------------------------------------------------------------------
// Semi-Circle
// This function receives one argument by reference and one by
// constant reference
void __fastcall MomentOfInertia(double& Moment, const double& R)
{
    const double PI = 3.14159;

    Moment = R * R * R * R * PI/ 8;
}
//---------------------------------------------------------------------------
// Triangle
// This function receives one argument by reference, two arguments by
// constant references and one argument by value
void __fastcall MomentOfInertia(double& Moment,
                      const double& b, const double& h, const int&)
{
    Moment = b * h * h * h / 12;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    double Length, Height, Radius, MRectangle, MSemiCircle, MTriangle;
    void __fastcall GetBaseAndHeight(double&, double&);
    void __fastcall GetRadius(double&);

    cout << "Enter the dimensions of the rectangle\n";
    GetBaseAndHeight(Length, Height);
    MomentOfInertia(MRectangle, Length, Height);
```

```
        cout << "Rectangle\n"
             << "Moment of inertia with regard to the X axis: ";
        cout << "I = " << MRectangle << "mm\n\n";
        cout << "Enter the radius of the semi-circle\n";

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

        GetBaseAndHeight(Length, Height);
        MomentOfInertia(MRectangle, Length, Height, 1);
        cout << "\nTriangle\n"
             << "Moment of inertia with regard to the X axis: ";
        cout << "I = " << MRectangle << "mm\n";

        cout << "\n\nPress any key to continue...";
        getchar();
        return 0;
    }
    //---------------------------------------------------------------------------
    // Passing arguments by reference
    void __fastcall GetBaseAndHeight(double& B, double& H)
    {
        cout << "Enter Base:   ";
        cin >> B;
        cout << "Enter Height: ";
        cin >> H;
    }
    //---------------------------------------------------------------------------
    void __fastcall GetRadius(double& R)
    {
        cout << "Enter Radius: ";
        cin >> R;
    }
    //---------------------------------------------------------------------------
```

2. Test the program and return to Bcb.

Inline Functions

When you call a function B() from function A(), function A() sends a request and must get to Function B(). This is sometimes cumbersome for long functions. Whenever your program includes a small function, C++ allows you to include such a function where it is being called. When function B() calls function A(), instead of sending a request to function A(), the compiler would include a copy of function A() into function B() where it is being called. Such a function (function A()) is qualified inline.

To create a function as inline, use the inline keyword when declaring the function as well as when defining it. Here is an example that makes use of an inline function:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop
//---------------------------------------------------------------------------
#pragma argsused
inline void Area(float Side)
{
    cout << "The area of the square is " << Side * Side;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    float s;
```



```
    cout << "Enter the side of the square: ";
    cin >> s;
    Area(s);

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

Here is an example of running the program:

```
Enter the side of the square: 14.55
The area of the square is 211.702

Press any key continue...
```

You can also use the __fastcall keyword on an inline function. To declare a function as inline and __fastcall, type both words at the beginning of the declaration. The following program requests the hourly salary from the user. Then it calculates the periodic earnings. The functions have been declared and defined as inline using the __fastcall technique:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop
//---------------------------------------------------------------------------
#pragma argsused
void inline __fastcall RequestSalary(double& h);
inline double __fastcall Daily(double h);
double inline __fastcall Weekly(double h);
inline double __fastcall BiWeekly(double h);
double inline __fastcall Monthly(double h);
double inline __fastcall Yearly(double h);
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    double HourlySalary;

    cout << "This program allows you to evaluate your salary "
         << "for different periods\n";

    RequestSalary(HourlySalary);

    cout << "\nBased on the hourly rate you supplied, here are your "
         << "periodic earnings";
    cout << "\n\tHourly:    $" << HourlySalary;
    cout << "\n\tDaily:     $" << Daily(HourlySalary);
    cout << "\n\tWeekly:    $" << Weekly(HourlySalary);
    cout << "\n\tBi-Weekly: $" << BiWeekly(HourlySalary);
    cout << "\n\tMonthly:   $" << Monthly(HourlySalary);
    cout << "\n\tYearly:    $" << Yearly(HourlySalary);

    cout << "\n\nPress any key continue...";
    getchar();
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
void inline __fastcall RequestSalary(double& x)
{
    cout << "Enter your hourly salary: $";
    cin >> x;
}
//---------------------------------------------------------------------------
inline double __fastcall Daily(double x)
{
```

```
    return x * 8;
}
//---------------------------------------------------------------------------
double inline __fastcall Weekly(double x)
{
    return Daily(x) * 5;
}
//---------------------------------------------------------------------------
inline double __fastcall BiWeekly(double x)
{
    return Weekly(x) * 2;
}
//---------------------------------------------------------------------------
double inline __fastcall Monthly(double x)
{
    return Weekly(x) * 4;
}
//---------------------------------------------------------------------------
double inline __fastcall Yearly(double h)
{
    return Monthly(h) *  12;
}
//---------------------------------------------------------------------------
```

Here is an example of running the program:

```
This program allows you to evaluate your salary for different periods
Enter your hourly salary: $15.55

Based on the hourly rate you supplied, here are your periodic earnings
        Hourly:    $15.55
        Daily:     $124.4
        Weekly:    $622
        Bi-Weekly: $1244
        Monthly:   $2488
        Yearly:    $29856

Press any key continue...
```

Using inline Functions

1. To use inline function in our application, change the MomentOfInertia() functions as follows:

```
//---------------------------------------------------------------------------
// Rectangle
// This function receives one argument by reference and two arguments
// by constant references
void inline __fastcall MomentOfInertia(double& Moment,
                    const double& b, const double& h)
{
    Moment = b * h * h * h / 3;
}
//---------------------------------------------------------------------------
// Semi-Circle
// This function receives one argument by reference and one by
// constant reference
void inline __fastcall MomentOfInertia(double& Moment, const double& R)
{
    const double PI = 3.14159;

    Moment = R * R * R * R * PI/ 8;
}
//---------------------------------------------------------------------------
// Triangle
// This function receives one argument by reference, two arguments by
// constant references and one argument by value
void inline __fastcall MomentOfInertia(double& Moment,
```



```
                    const double& b, const double& h, const int&)
{
    Moment = b * h * h * h / 12;
}
//---------------------------------------------------------------------------
```

2. Test the program:

```
Enter the dimensions of the rectangle
Enter Base:   12.12
Enter Height: 12.12
Rectangle
Moment of inertia with regard to the X axis: I = 7192.65mm

Enter the radius of the semi-circle
Enter Radius: 12.12
Semi-Circle
Moment of inertia with regard to the X axis: I = 8473.64mm

Enter the dimensions of the triangle
Enter Base:   12.12
Enter Height: 12.12

Triangle
Moment of inertia with regard to the X axis: I = 1798.16mm


Press any key to continue...
```

3. Return to Bcb

Static Variables

Consider the following program:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop
//---------------------------------------------------------------------------
#pragma argsused
//---------------------------------------------------------------------------
void __fastcall Starter(int y)
{
    double a = 112.50;
    double b = 175.25;

    a = a / y;
    b = b + 2;

    cout << "y = " << y << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "b / a = " << b / a << "\n\n";
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    Starter(2);
    Starter(2);
    Starter(2);
    Starter(2);

    cout << "Press any key continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

When executed, this program would produce:

```
y = 2
a = 56.25
b = 177.25
b / a = 3.15111

y = 2
a = 56.25
b = 177.25
b / a = 3.15111

y = 2
a = 56.25
b = 177.25
b / a = 3.15111

y = 2
a = 56.25
b = 177.25
b / a = 3.15111

Press any key continue...
```

The Starter() function receives one argument passed when it is called. The called function also receives the same argument everytime. Looking at the result, the argument passed to the function and the local variables declared inside of the called function keep the same value everytime the function is called. That is, when the Starter() function is exited, the values remain the same.

We know that, when a function is defined, any variable declared locally belongs to the function and its influence cannot expand beyond the presence of the function. If you want a locally declared variable to keep its changed value when its host function is exited, declare such a variable as static.

To declare a static variable, type the keyword static on the left of the variable's data type. For example, if you plan to declare a Radius variable as static in an Area() function, you could write:

```
//---------------------------------------------------------------------------
double __fastcall Area()
{
    static double Radius;
}
//---------------------------------------------------------------------------
```

You should always initialize a static variable before using it; that is, when declaring it. To make the local variables of our Starter() function static, we can declare them as follows:

```
//---------------------------------------------------------------------------
void __fastcall Starter(int y)
{
    static double a = 112.50;
    static double b = 175.25;

    a = a / y;
    b = b + 2;

    cout << "y = " << y << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "b / a = " << b / a << "\n\n";
}
//---------------------------------------------------------------------------
```

This time, when executing the program, it would produce:

```
y = 2
```



```
a = 56.25
b = 177.25
b / a = 3.15111

y = 2
a = 28.125
b = 179.25
b / a = 6.37333

y = 2
a = 14.0625
b = 181.25
b / a = 12.8889

y = 2
a = 7.03125
b = 183.25
b / a = 26.0622

Press any key continue...
```

Notice that, this time, each local variable keeps its newly changed value when the function exits. Since a function's argument can receive different values as the function is called different times, we can test our program by passing different values to its argument as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop
//---------------------------------------------------------------------------
#pragma argsused
//---------------------------------------------------------------------------
void __fastcall Starter(int y)
{
    static double a = 112.50;
    static double b = 175.25;

    a = a / y;
    b = b + 2;

    cout << "y = " << y << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "b / a = " << b / a << "\n\n";

}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    Starter(2);
    Starter(5);
    Starter(14);
    Starter(25);

    cout << "Press any key continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

The current version of the program would produce:

```
y = 2
a = 56.25
b = 177.25
b / a = 3.15111

y = 5
a = 11.25
```

```
b = 179.25
b / a = 15.9333

y = 14
a = 0.803571
b = 181.25
b / a = 225.556

y = 25
a = 0.0321429
b = 183.25
b / a = 5701.11

Press any key continue...
```

Creating Files

When writing a program, the main reason for using functions is to isolate assignments. This allows you to effectively troobleshoot problems when they arise. For example, if you are asked to write a program that would process orders at a department store, you can write one long main() function that would process all requests and orders. When the store is having a sale and you need to apply a discount to the program, you would spend time looking for the sections that would use the discount and calculate the price. If you use functions to isolate assignments, you can easily find out which particular function deals with discount; all you would have to do is change the discount value without having to read the whole program.

When using functions in a program, we found out that the order of declaring functions was important. For example, you cannot call a function that has not been declared yet. For this reason, whenever you need to call a function, you should find out where and whether it has been declared already. If the program is using many functions, it would become coumbersome to start looking for functions, although the Class Explorer can help you. At the same time, on a large program, it is usual for many functions to use the same kind of variable. Although you can locally declare the same variable needed by a function, if these functions of the same program would need to exchange values among them, you should declare some variables globally, usually on top of the file, then make such a variable available to any function that needs it.

To make these functions and variables easily manageable, you can create one file where you would list the functions and variables used in a program. Such a file is called a header file and it has the .h extension. By default, a newly created header file is called File1.h. If you create additional header files, they would have incremental names such as File2.h, File3.h, etc. If you want to change the name of a header file, you must save it and rename it.

To create a header file, from the New property page of the New Items dialog box, select the Header File icon and declare the variables and functions. Here is an example of a header file:

```
void RequestItemName(char Name[40]);
void RequestOrigPrice(double& Original);
void RequestDiscountRate(double& Discount);
void RequestTaxRate(double& Tax);
void CalcDiscount(double Price, double Discount);
void CalcTaxAmount(double Price, double Tax);
void CalcNetPrice(double Orig, double DiscAmt, double TaxAmt);
void ProcessTheOrder();
void ShowReceipt();
```

After creating the header file, you can create another file to define them. The file used to implement the functions of the header file is called the Source File and it has a .cpp extension. By default, the first source file is called File1.cpp. If you

create additional files, they would have incremental names such as File2.cpp, File3.cpp, etc. If you want to change a source file name, you must save it and rename it.

To create a source file, from the New property page of the New Items dialog box, select the Cpp File icon and implement the functions.

When implementing the functions declared in a header file, you must provide the name(s) of the header file(s) where the function(s) is(are) declared. For example, if a function declared as void ReviewApplication() is in a header file called Membership.h, to use such a function, you must type #include “Membership.h” on top of the source file. Here is the source file of the functions declared above:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#include <stdio.h>
// Used to request a string from the user
#include "File1.h"
char ItemName[40];
double OriginalPrice;
double DiscountRate;
double DiscountAmount;
double TaxRate;
double TaxAmount;
double NetPrice;
//---------------------------------------------------------------------------
void __fastcall RequestItemName(char Name[])
{
    cout << " - K & J Department Store -\n” << “Enter the items name: ";
    gets(Name);
}
//---------------------------------------------------------------------------
void __fastcall RequestOrigPrice(double& o)
{
    cout << "Enter the original price: $";
    cin >> o;
}
//---------------------------------------------------------------------------
void __fastcall RequestDiscountRate(double& d)
{
    cout << "Enter discount rate: ";
    cin >> d;
}
//---------------------------------------------------------------------------
void __fastcall RequestTaxRate(double& t)
{
    cout << "Enter the tax rate: ";
    cin >> t;
}
//---------------------------------------------------------------------------
void __fastcall CalcDiscount(double Price, double Discount)
{
    DiscountAmount = Price * DiscountRate / 100;
}
//---------------------------------------------------------------------------
void __fastcall CalcTaxAmount(double Price, double Tax)
{
    TaxAmount = Price * TaxRate / 100;
}
//---------------------------------------------------------------------------
void __fastcall CalcNetPrice(double Price, double Discount, double Tax)
{
    NetPrice = Price - Discount + Tax;
}
//---------------------------------------------------------------------------
void __fastcall ProcessTheOrder()
{
    RequestItemName(ItemName);
    RequestOrigPrice(OriginalPrice);
```

```
    RequestDiscountRate(DiscountRate);
    RequestTaxRate(TaxRate);
    CalcDiscount(OriginalPrice, DiscountAmount);
    CalcTaxAmount(OriginalPrice, TaxAmount);
    CalcNetPrice(OriginalPrice, DiscountAmount, TaxAmount);
}
//---------------------------------------------------------------------------
void __fastcall ShowReceipt()
{
    cout << "\nItem Name: " << ItemName;
    cout << "\nOriginal Price: $" << OriginalPrice;
    cout << "\nDiscount Amount: $" << DiscountAmount;
    cout << "\nTax Amount: $" << TaxAmount;
    cout << "\nNet Price: $" << NetPrice;
}
//---------------------------------------------------------------------------
```

With the functions declared and defined, you can call them from any section of the program, as long as you include the header file; you do not need to include the source file. The header file contains all the information the calling functions need. Here is the main() function calling the functions defined in the header file above:

```
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    ProcessTheOrder();
    ShowReceipt();

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

Here is a result of running the program:

```
Enter the items name: Len Gouland Suit
Enter the original price: $540.55
Enter discount rate: 40
Enter the tax rate: 7.55

Item Name: Len Gouland Suit
Original Price: $540.55
Discount Amount: $216.22
Tax Amount: $40.8115
Net Price: $365.142

Press any key to continue...
```

To make sure that the header file has not been created anywhere in the program, you should ask the compiler to check it. This is done using the #ifndef preprocessor followed by a one-word name for the file. Once the compiler has made sure that the header file is unique, you can ask the compiler to define it. At the end of the file (that is, when the components of the file have been declared), signal the closing of the file with the #endif preprocessor.

The variables that we used to perform our calculations were declared in the source file (File1.cpp). The processing of the order is defined by the header file. For this reason, the variables declared in the source would not be accessible to any function outside of the File1.cpp even if the calling file includes the File1.h. For example, the following implementation of the main() function would cause an error because the main() function is trying to call the OriginalPrice variable which is declare in the source file:

```
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
```



```
{
    ProcessTheOrder();
    ShowReceipt();

    cout << "\nOriginal Price: " << OriginalPrice;

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

The error would display as follows:

[C++ Error] Unit1.cpp(93): E2451 Undefined symbol 'OriginalPrice'

To make the variables of the previous header file accessible to any function that would include the File1.h header file, you must declare the variables in the header file. If you declare these variables using the same syntax of the source file, the compiler would throw a warning for each variable. Therefore, the variables should be declared as static. Here is the new version of our header file:

```
#ifndef DISCOUNTER_H
#define DISCOUNTER_H

static char ItemName[40];
static double OriginalPrice;
static double DiscountRate;
static double DiscountAmount;
static double TaxRate;
static double TaxAmount;
static double NetPrice;

void __fastcall RequestItemName(char Name[40]);
void __fastcall RequestOrigPrice(double& Original);
void __fastcall RequestDiscountRate(double& Discount);
void __fastcall RequestTaxRate(double& Tax);
void __fastcall CalcDiscount(double Price, double Discount);
void __fastcall CalcTaxAmount(double Price, double Tax);
void __fastcall CalcNetPrice(double Orig, double DiscAmt, double TaxAmt);
void __fastcall ProcessTheOrder();
void __fastcall ShowReceipt();

#endif // Closing DISCOUNTER_H
```

Here is the new source file:

```
#include <iostream.h>
#include <stdio.h>
// Used to request a string from the user
#include "File1.h"
//---------------------------------------------------------------------------
void __fastcall RequestItemName(char Name[])
{
    cout << "Enter the items name: ";
    gets(Name);
}
//---------------------------------------------------------------------------
void __fastcall RequestOrigPrice(double& o)
{
    cout << "Enter the original price: $";
    cin >> o;
}
//---------------------------------------------------------------------------
void __fastcall RequestDiscountRate(double& d)
{
    cout << "Enter discount rate: ";
    cin >> d;
}
```

```
//---------------------------------------------------------------------------
void __fastcall RequestTaxRate(double& t)
{
    cout << "Enter the tax rate: ";
    cin >> t;
}
//---------------------------------------------------------------------------
void __fastcall CalcDiscount(double Price, double Discount)
{
    DiscountAmount = Price * DiscountRate / 100;
}
//---------------------------------------------------------------------------
void __fastcall CalcTaxAmount(double Price, double Tax)
{
    TaxAmount = Price * TaxRate / 100;
}
//---------------------------------------------------------------------------
void __fastcall CalcNetPrice(double Price, double Discount, double Tax)
{
    NetPrice = Price - Discount + Tax;
}//---------------------------------------------------------------------------
void __fastcall ProcessTheOrder()
{
    RequestItemName(ItemName);
    RequestOrigPrice(OriginalPrice);
    RequestDiscountRate(DiscountRate);
    RequestTaxRate(TaxRate);
    CalcDiscount(OriginalPrice, DiscountAmount);
    CalcTaxAmount(OriginalPrice, TaxAmount);
    CalcNetPrice(OriginalPrice, DiscountAmount, TaxAmount);
}
//---------------------------------------------------------------------------
void __fastcall ShowReceipt()
{
    cout << "\nItem Name: " << ItemName;
    cout << "\nOriginal Price: $" << OriginalPrice;
    cout << "\nDiscount Amount: $" << DiscountAmount;
    cout << "\nTax Amount: $" << TaxAmount;
    cout << "\nNet Price: $" << NetPrice;
}
//---------------------------------------------------------------------------
```

Here is the implementation of the main() function:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop'
#include "File1.h"
//---------------------------------------------------------------------------
#pragma argsused
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    ProcessTheOrder();
    ShowReceipt();

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

Here is an example of running the program:

```
Enter the items name: Lamy Jeans
Enter the original price: $45.95
Enter discount rate: 35
Enter the tax rate: 7.55

Item Name: Lamy JeansOriginal Price: $45.95
```

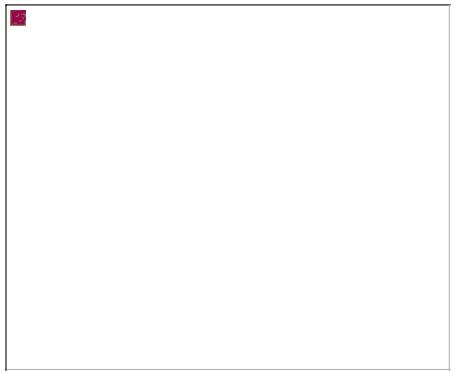


```
Discount Amount: $16.0825
Tax Amount: $3.46923
Net Price: $33.3367

Press any key to continue...
```

Creating Files

1. Start a new application. Create it using the Console Wizard and make sure you select the C++ radio button in the Console Wizard dialog box.
2. To save the project, on the Standard toolbar, click the Save All button
3. Click the arrow of the Save In combo box and locate the My Documents folder
4. Click the Create New Folder button
5. Type Inertia and press Enter. Double-click Inertia to display it in the Save In combo box.
6. Replace the name Unit1 with Main and make sure that the Save As Type combo box is displaying C++Builder Unit (*.cpp).
7. Press Enter
8. Replace the name of the project with Inertie and press Enter.
9. To create a header file, on the main menu of the C++ Builder, click File -> New...
10. From the New property page of the New Items dialog box, click the Header File icon:



11. Click OK
12. To save the header file, on the main menu, click File -> Save
13. In the File Name edit box, type Inertia.h and make sure you include the extension:

14. Click Save
15. In the empty file, type:

```
#ifndef Inertia_h
#define Inertia_h
//---------------------------------------------------------------------------
double __fastcall MomentOfInertia(const double b, const double h);
double __fastcall MomentOfInertia(const double R);
double __fastcall MomentOfInertia(const double b, const double h, const int);
double __fastcall GetBase();
double __fastcall GetHeight();
double __fastcall GetRadius();
//---------------------------------------------------------------------------
#endif // Inertia_h
```

16. To create the associated source file, on the main menu, click File -> New...
17. From the New property page of the New Items dialog box, click the Cpp File icon:

18. To save the source file, on the main menu, click File -> Save
19. In the Save File1 As dialog box, replace the name of the file with Inertia.cpp
20. Click Save
21. Replace the empty file with:

```
#include <iostream.h>
#include "Inertia.h"
//---------------------------------------------------------------------------
```



```
// Rectangle
double __fastcall MomentOfInertia(const double b, const double h)
{
    return b * h * h * h / 3;
}
//---------------------------------------------------------------------------
// Semi-Circle
double __fastcall MomentOfInertia(const double R)
{
    const double PI = 3.14159;
    return R * R * R * R * PI/ 8;
}
//---------------------------------------------------------------------------
// Triangle
double __fastcall MomentOfInertia(const double b, const double h, const int)
{
    return b * h * h * h / 12;
}
//---------------------------------------------------------------------------
double __fastcall GetBase()
{
    double B;

    cout << "Base: ";
    cin >> B;

    return B;
}
//---------------------------------------------------------------------------
double __fastcall GetHeight()
{
    double H;

    cout << "Height: ";
    cin >> H;

    return H;
}
//---------------------------------------------------------------------------
double __fastcall GetRadius()
{
    double R;

    cout << "Radius: ";
    cin >> R;

    return R;
}
//---------------------------------------------------------------------------
```

22. Click the Main.cpp tab to access the main() function.
23. Change the file as follows:

```
//---------------------------------------------------------------------------
#include <iostream.h>
#pragma hdrstop
#include "Inertia.h"
//---------------------------------------------------------------------------
#pragma argsused
void inline Announce(const string Figure)
{
    cout << "Enter the dimensions of the " << Figure << "\n";
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    double Length, Height, Radius;

    Announce("rectangle");
    Length = GetBase();
```

```
        Height = GetHeight();
        cout << "Rectangle\n"
             << "Moment of inertia with regard to the X axis: ";
        cout << "I = " << MomentOfInertia(Length, Height) << "mm\n\n";

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

        Announce("triangle");
        Length = GetBase();
        Height = GetHeight();

        cout << "\nTriangle\n"
             << "Moment of inertia with regard to the X axis: ";
        cout << "I = " << MomentOfInertia(Length, Height, 1) << "mm\n";

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

24. To save the project, on the main menu, click file -> Save All
25. To test the program, on the Debug toolbar, click the Run button

Previous

Next