Introduction to Classes

Classes: An Introduction

The data types we have applied so far to our variables were used to identify individual items. To create more advanced and complete objects, C++ allows you to group these identifiers and create a newly defined object.

Introduction to Classes

An object, such as a CD Player, a printer, a car, etc, is built from assembling various parts. In the same way, C++ allows you to group various variables and create a new object called a class.

Imagine a company that manufactures shoe boxes hires you to write a program that would help design and identify those shoe boxes. A shoe box is recognized for its dimensions (length, width, height), color, and shoe size that a particular box can contain, etc. The variables that characterize such an object could be:

Double Length, Width, Height;
PChar Color;
Single ShoeSize;

And the program that defines a shoe box object could be:

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

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

#pragma argsused
int main(int argc, char* argv[])
{
    // Define the characteristics of a shoe box
    // The following characteristics are COMPLETELY random
    Double Length(12.55), Width(6.32), Height(8.74);
    PChar Color("Yellow Stone");
    Single ShoeSize = 10.50;

    // Display the characteristics of the shoe box
    cout << "Characteristics of this shoe box";
    cout << "\n\tLength = " << Length
         << "\n\tWidth = " << Width
         << "\n\tHeight = " << Height
         << "\n\tVolume = " << Length * Width * Height
         << "\n\tColor = " << Color
         << "\n\tSize = " << ShoeSize;

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

The program would produce:

```
Characteristics of this shoe box
        Length = 12.55
        Width = 6.32
        Height = 8.74
        Volume = 693.222
        Color = Yellow Stone
        Size = 10.5

Press any key to continue...
```


Unless dealing with one shoe box, this program would be rudimentary to run for each object. The solution is to create an object called box that groups everything that characterizes the object.

Object Concept

1. Start Bcb if not yet. To create a new project, on the main menu, click File -> New...
2. From the New property sheet of the New Items dialog box, click the Console Wizard icon and click OK.
3. On the Console Wizard dialog, click the C++ radio button and the Console Application check box only (no VCL, no Multi-Threaded).
4. Click OK
5. To save the project, on the Standard toolbar, click the Save All button.
6. Create a new folder called Employees
7. Save the file as Main in the Employees folder
8. Save the project as Employees
9. Change the content of the file as follows:

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

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

#pragma argsused
int main(int argc, char* argv[])
{
    string FirstName = "Bertine";
    string LastName = "Lamond";
    double TotalHours = 36.50;
    double HourlySalary = 8.52;
    double WeeklySalary = TotalHours * HourlySalary;

    cout << "Information about the employee";
    cout << "\n\tEmployee Name: " << FirstName << " " << LastName;
    cout << "\n\tWeekly Hours: " << TotalHours;
    cout << "\n\tHourly Salary: $" << HourlySalary;
    cout << "\n\tWeekly Salary: $" << WeeklySalary;

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

10. To test the program, press F9. The program would produce:

```
Information about the employee
        Employee Name: Bertine Lamond
        Weekly Hours: 36.5
        Hourly Salary: $8.52
        Weekly Salary: $310.98
```

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

11. Return to Bcb
12. To request the values of the variables, change the program as follows:

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

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

#pragma argsused
int main(int argc, char* argv[])
{
    string FirstName, LastName;
    double TotalHours, HourlySalary;
    double WeeklySalary;

    cout << "Enter the following pieces of information about the employee\n";
    cout << "First Name: ";
    cin >> FirstName;
    cout << "Last Name: ";
    cin >> LastName;
    cout << "Hours worked this week: ";
    cin >> TotalHours;
    cout << "Hourly Salary: $";
    cin >> HourlySalary;

    WeeklySalary = TotalHours * HourlySalary;

    cout << "\nInformation about the employee";
    cout << "\n\tEmployee Name: " << FirstName << " " << LastName;
    cout << setiosflags(ios::fixed) << setprecision(2);
    cout << "\n\tWeekly Hours: " << TotalHours;
    cout << "\n\tHourly Salary: $" << HourlySalary;
    cout << "\n\tWeekly Salary: $" << WeeklySalary;

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

13. Press F9 to test the program:

```
Enter the following pieces of information about the employee
First Name: Alain
Last Name: Browns
Hours worked this week: 35.50
Hourly Salary: $12.20

Information about the employee
        Employee Name: Alain Browns
        Weekly Hours: 35.50
        Hourly Salary: $12.20
        Weekly Salary: $433.10

Press any key to continue...
```

14. Return to Bcb

Creating a Class

To create a class, use the class keyword followed by a name for the object. Like any other declared variable, the class declaration ends with a semi-colon. The name of a class follows the rules we have applied so far for variable and function names. To

declare a class called ShoeBox, we would type the following:

```
class ShoeBox;
```

As a name that represents a group of items, a class has a body that would be used to define the items that compose it. The body of a class starts with an opening curly bracket "{" and ends with a closing one "}". Therefore, another way to create a class is:

```
class ClassName{};
```

This could also be created as:

```
class ClassName {
};
```

or

```
class ClassName
{
};
```

Either of these techniques produces the same effect.

Since a class is built from combining other identifiers, you will list each variable inside of the body of the class. Each item that composes the class is represented as a complete variable declared with a data type and a name. As a variable, each declaration must end with a semi-colon.

Continuing with our shoe box object, we could create it using the class as follows:

```
class ShoeBox
{
    Double Length, Width, Height;
    PChar Color;
    Single ShoeSize;
};
```

The items that compose a class are called members of the class. As a convention, the names of objects in C++ Builder start with T. For example, an object created as

```
class Parking;
```

would be created as

```
class TParking;
```

This naming convention applies to enumerators, structures, unions, and classes. We will also use it when naming our objects.

Creating a Class

1. To add a class to our exercise, change the content of the file as follows:

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

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

#pragma argsused
class TEmployee
{
    string FirstName;
    string LastName;
```

```
    double TotalHours;
    double HourlySalary;
    double WeeklySalary;
};
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    cout << "Press any key to continue...";
    getchar();
    return 0;
}
//---------------------------------------------------------------------------
```

2. To test the program, on the main menu, click Run -> Run
3. As you can see, the program does not do much. Press any key to return to Bcb

Accessing a Class

A common object in real life is visibly made of two categories of parts: those you can see or touch and those you do not have access to. The parts you can see or touch are considered visible or accessible. In C++, such parts are referred to as public. Those you cannot see or touch are considered hidden. In C++, such parts are referred to as private. Like objects in real life, a class is made of sections that the other functions or other objects cannot "see" and those the other objects can access. The other objects of of the program are sometimes referred to as the clients of the object. The parts the client of an object can touch in a class are considered public and the others are private.

When creating a class, you will define which items are public and which ones are private. The items that are public are created in a section that starts with the public keyword followed by a semi-colon. The others are in the private section. If you do not specify these sections, all of the members of a class are considered private. For example, all of the members of the previously defined TShoeBox class are private.

Using the public and private sections, our shoe box object can be created as:

```
class TShoeBox
{
public:
    Double Length, Width, Height;
    PChar Color;
private:
    Single ShoeSize;
};
```

The public and private keywords are referenced by their access level because they control how much access a variable allows. You can create as many public sections or as many private sections as you want. For example, the above class could be created as:

```
class TShoeBox
{
public:
    Double Length, Width, Height;
public:
    PChar Color;
    Double Volume;
private:
    Single ShoeSize;
private:
    Char Material;
    String Color;
};
```

When creating a class with different public and private sections, all of the declared

variables under an access level keyword abide by the rules of that access level. The fact that you use different public sections does not by any means warrant different public levels to the variables. A variable declared as public in one public section has the same public level of access as any other variable that is declared in another public section.

A variable that is declared in a class is called a member of the class. Once the class has been defined, you can use it as an individual variable.

Accessing a Class

- To add access levels to our class, change the class definition as follows:

```
//---------------------------------------------------------------------------
class TEmployee
{
public:
    string FirstName;
    string LastName;
    double TotalHours;
    double HourlySalary;
private:
};
//---------------------------------------------------------------------------
```

Declaring a Class

After defining a class, you can declare it as a variable using the same syntax we have used for any other variable. A class is declared using its name followed by a name for the defined variable and ending with a semi-colon. For example, our TShoeBox class can be declared as follows:

TShoeBox Shake;

When an object has been declared, you can access any of its members using the member access operator ".". First, type the name of the object variable, followed by a period, followed by the name of the member you want to access. For example, to access the member Length of the above class, you would write:

Shake.Length;

Using this syntax, you can display the value of a class member:

cout << Shake.Length;

or you can request its value from the user, using the cin operator. Here is an example:

cin >> Shake.Lengh;

C++ Builder can help you remember and select the right member of an object. When using an object variable, type the declared variable name followed by a period. C++ Builder will display the list of accessible members; you can choose from the list. Type the first letter of the member and it would be selected. If there is more than one member that start with the same letter, you can continue typing subsequent letters or you can scroll in the list. You can also click the desired member in the list. Once the item is highlighted, press the Spacebar and continue your work. This feature is referred to as Code Completion (in some Microsoft documents, it is called Intellisense). Sometimes, the code completion seems slow. You can modify its timing in the Code Insight property page of the Editor Properties dialog box:

By default, the Delay timer is set to 1 second. I recommend you set it to the lowest value:

Using the cout operator to display the values of the object members, our program could be as follows:

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

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

#pragma argsused
class TShoeBox{
public:
    Double Length, Width, Height;
    PChar Color;
private:
    Single ShoeSize;
};
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TShoeBox Shake;

    // Display the characteristics of the shoe box
    cout << "Characteristics of this shoe box";
    cout << "\n\tLength = " << Shake.Length
         << "\n\tWidth = " << Shake.Width
         << "\n\tHeight = " << Shake.Height
         << "\n\tVolume = " << Shake.Length * Shake.Width * Shake.Height
         << "\n\tColor = " << Shake.Color
         << "\n\tSize = " << Shake.ShoeSize;

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



```
//---------------------------------------------------------------------------
```

At this time, because of trying to access a private member, the program would produce the following error

[C++ Error] Unit1.cpp(30): E2247 'TShoeBox::TShoeSize' is not accessible

Even if you change the ShoeSize member access from private to public, the program would render unpredictable results because the members have not been given appropriate values:

Characteristics of this shoe box Length = 0 Width = 1.79571e-307 Height = 4.17266e-315 Volume = 0 Color = Size = 3.58732e-43Press any key to continue...

Declaring a Class

1. Declare an employee object and initialize its members as follows:

```
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TEmployee FullTime;
    double WeeklySalary;

    FullTime.FirstName = "Chester";
    FullTime.LastName = "Stanley";
    FullTime.TotalHours = 42.00;
    FullTime.HourlySalary = 10.63;

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

2. Display the members of the Employee class as follows:

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

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

#pragma argsused
class TEmployee
{
public:
    string FirstName;
    string LastName;
    double TotalHours;
    double HourlySalary;
    double WeeklySalary;
private:
};
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TEmployee FullTime;
    double WeeklySalary;
```

```
    FullTime.FirstName = "Chester";
    FullTime.LastName = "Stanley";
    FullTime.TotalHours = 42.00;
    FullTime.HourlySalary = 10.63;

    WeeklySalary = FullTime.TotalHours * FullTime.HourlySalary;

    cout << "Information about the employee";
    cout << "\n\tFull Name: "
         << FullTime.FirstName << " " << FullTime.LastName;
    cout << "\n\tTotal Weekly Hours: "
         << setiosflags(ios::fixed) << setprecision(2)
         << FullTime.TotalHours;
    cout << "\n\tHourly Salary: $" << FullTime.HourlySalary;
    cout << "\n\tWeekly Salary: $" << WeeklySalary;

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

3. Press F9 to test the program:

```
Information about the employee
        Full Name: Chester Stanley
        Total Weekly Hours: 42.00
        Hourly Salary: $10.63
        Weekly Salary: $446.46

Press any key to continue...
```

4. Return to Bcb
5. To request the values of the members of a class, change the file as follows:

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

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

#pragma argsused
class TEmployee
{
public:
    string FirstName;
    string LastName;
    double TotalHours;
    double HourlySalary;
    double WeeklySalary;
private:
};
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TEmployee Seasonal;
    double WeeklySalary;

    cout << "Enter the following pieces of information about this employee\n";
    cout << "First Name: ";
    cin >> Seasonal.FirstName;
    cout << "Last Name: ";
    cin >> Seasonal.LastName;
    cout << "Hourly Salary: $";
    cin >> Seasonal.HourlySalary;
    cout << "Total Weekly Hours: ";
    cin >> Seasonal.TotalHours;
```



```
    WeeklySalary = Seasonal.TotalHours * Seasonal.HourlySalary;

    cout << "\nInformation about the employee";
    cout << "\n\tFull Name: "
         << Seasonal.FirstName << " " << Seasonal.LastName;
    cout << "\n\tTotal Weekly Hours: "
         << setiosflags(ios::fixed) << setprecision(2)
         << Seasonal.TotalHours;
    cout << "\n\tHourly Salary: $" << Seasonal.HourlySalary;
    cout << "\n\tWeekly Salary: $" << WeeklySalary;

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

6. Press F9 to test the program:

```
Enter the following pieces of information about this employee
First Name: Marlyse
Last Name: Dietch
Hourly Salary: $10.85
Total Weekly Hours: 42.50

Information about the employee
        Full Name: Marlyse Dietch
        Total Weekly Hours: 42.50
        Hourly Salary: $10.85
        Weekly Salary: $461.12

Press any key to continue...
```

7. Return to Bcb
8. To calculate the value of a member using values from external variables, change the file as follows:

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

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

#pragma argsused
class TEmployee
{
public:
    string FirstName;
    string LastName;
    double TotalHours;
    double HourlySalary;
    double WeeklySalary;
private:
};
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TEmployee Seasonal;
    // Hours for each day
    double Mon, Tue, Wed, Thu, Fri, Sat, Sun;
    double WeeklySalary;

    cout << "Enter the following pieces of information about this employee\n";
    cout << "First Name: "; cin >> Seasonal.FirstName;
    cout << "Last Name: "; cin >> Seasonal.LastName;
    cout << "Hourly Salary: $"; cin >> Seasonal.HourlySalary;
    cout << "Enter the number of hours for each day\n";
```

```
    cout << "Monday: "; cin >> Mon;
    cout << "Tuesday: "; cin >> Tue;
    cout << "Wednesday: "; cin >> Wed;
    cout << "Thursday: "; cin >> Thu;
    cout << "Friday: "; cin >> Fri;
    cout << "Saturday: "; cin >> Sat;
    cout << "Sunday: "; cin >> Sun;

    Seasonal.TotalHours = Mon + Tue + Wed + Thu + Fri + Sat + Sun;
    WeeklySalary = Seasonal.TotalHours * Seasonal.HourlySalary;

    cout << "\nInformation about the employee";
    cout << "\n\tFull Name: "
         << Seasonal.FirstName << " " << Seasonal.LastName;
    cout << "\n\tTotal Weekly Hours: "
         << setiosflags(ios::fixed) << setprecision(2)
         << Seasonal.TotalHours;
    cout << "\n\tHourly Salary: $" << Seasonal.HourlySalary;
    cout << "\n\tWeekly Salary: $" << WeeklySalary;

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

9. Press F9 to test the program. Here is an example:

```
Enter the following pieces of information about this employee
First Name: Theodore
Last Name: Amis
Hourly Salary: $8.95
Enter the number of hours for each day
Monday: 8.00
Tuesday: 8.50
Wednesday: 9.00
Thursday: 8.00
Friday: 8.00
Saturday: 0
Sunday: 0

Information about the employee
        Full Name: Theodore Amis
        Total Weekly Hours: 41.50
        Hourly Salary: $8.95
        Weekly Salary: $371.42

Press any key to continue...
```

10. Return to Bcb
11. When the user is typing the amount of hours for each day, you should make sure that entries such –8.00 or 25 are not allowed because the user cannot work for negative hours, and the user cannot work more than 24 hour in one day. To take care of this situation, implementation a function that would use a conditional statement to control what the user would type. Consequently, you can change the implementation of the main() function:

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



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

#pragma argsused
class TEmployee
{
```



```
public:
    string FirstName;
    string LastName;
    double TotalHours;
    double HourlySalary;
    double WeeklySalary;
private:
};
//---------------------------------------------------------------------------
double __fastcall ObtainDailyHours(string s)
{
    double h;

    do {
        cout << s << ": ";
        cin >> h;
        if(h < 0 || h > 24)
            cout << "Please enter a number between 0.00 and 24.00\n";
    }while(h < 0 || h > 24);

    return h;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TEmployee Seasonal;
    // Hours for each day
    double Mon, Tue, Wed, Thu, Fri, Sat, Sun;
    double WeeklySalary;

    cout << "Enter the following pieces of information about this employee\n";
    cout << "First Name: "; cin >> Seasonal.FirstName;
    cout << "Last Name: "; cin >> Seasonal.LastName;
    cout << "Hourly Salary: $"; cin >> Seasonal.HourlySalary;
    cout << "Enter the number of hours for each day\n";
    Mon = ObtainDailyHours("Monday");
    Tue = ObtainDailyHours("Tuesday");
    Wed = ObtainDailyHours("Wednesday");
    Thu = ObtainDailyHours("Thursday");
    Fri = ObtainDailyHours("Friday");
    Sat = ObtainDailyHours("Saturday");
    Sun = ObtainDailyHours("Sunday");

    Seasonal.TotalHours = Mon + Tue + Wed + Thu + Fri + Sat + Sun;
    WeeklySalary = Seasonal.TotalHours * Seasonal.HourlySalary;

    cout << "\nInformation about the employee";
    cout << "\n\tFull Name: "
         << Seasonal.FirstName << " " << Seasonal.LastName;
    cout << "\n\tTotal Weekly Hours: "
         << setiosflags(ios::fixed) << setprecision(2)
         << Seasonal.TotalHours;
    cout << "\n\tHourly Salary: $" << Seasonal.HourlySalary;
    cout << "\n\tWeekly Salary: $" << WeeklySalary;

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

12. Test the program to make sure it is working. Here is an example:

```
Enter the following pieces of information about this employee
First Name: Paul
Last Name: Kitts
Hourly Salary: $10.25
Enter the number of hours for each day
Monday: 8.00
Tuesday: -5
```

```
Please enter a number between 0.00 and 24.00
Tuesday: 8.00
Wednesday: 9.50
Thursday: 8.00
Friday: 8
Saturday: 0
Sunday: 0

Information about the employee
        Full Name: Paul Kitts
        Total Weekly Hours: 41.50
        Hourly Salary: $10.25
        Weekly Salary: $425.38

Press any key to continue...
```

13. Return to Bcb
14. Whenever doing a payroll, something you should always pay attention is the calculation of overtime if the employee has worked more than 40 hours in a week.
    Add the following commented function to take care of calculating weekly gross pay in factor of possible overtime:

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

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

#pragma argsused
class TEmployee
{
public:
    string FirstName;
    string LastName;
    double TotalHours;
    double HourlySalary;
    double WeeklySalary;
private:
};
//---------------------------------------------------------------------------
double __fastcall ObtainDailyHours(string s)
{
    double h;

    do {
        cout << s << ": ";
        cin >> h;
        if(h < 0 || h > 24)
            cout << "Please enter a number between 0.00 and 24.00\n";
    }while(h < 0 || h > 24);

    return h;
}
//---------------------------------------------------------------------------
double __fastcall CalcGrossPay(const double Hours, const double Salary)
{
    double Gross;

    if( Hours > 40 )
    {
        double OvertimeHours, RegularSalary, OvertimeGross;

        // If the total of weekly hours is above 40
        // then the employee has overtime
        RegularSalary = Salary * 40;

        // The overtime is anything above 40 hours
```



```
        OvertimeHours = Hours - 40;

        // Hourly overtime salary is calculated by adding half
        // of the salary to the regular hourly salary
        double OvertimeSalary = Salary + (Salary * 0.50);

        // To calculate the overtime salary, multiply the overtime
        // hours to the overtime salary
        OvertimeGross = OvertimeHours * OvertimeSalary;

        // Now we have the regular salary and the overtime wage
        // Simply add them to get the gross weekly earnings
        Gross = RegularSalary + OvertimeGross;
    }
    // Since the employee's total hours are less than 40,
    // there is no overtime
    else
    {
        //OvertimeHours = 0.00;
        //OvertimeGross = 0.00;
        //RegularHours = Hours;
        //RegularSalary = Salary * Hours;
        Gross = Salary * Hours;
    }

    return Gross;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TEmployee Seasonal;
    // Hours for each day
    double Mon, Tue, Wed, Thu, Fri, Sat, Sun, Total;
    double Hourly;//, WeeklySalary;

    cout << "Enter the following pieces of information about this employee\n";
    cout << "First Name: "; cin >> Seasonal.FirstName;
    cout << "Last Name: "; cin >> Seasonal.LastName;
    cout << "Hourly Salary: $"; cin >> Seasonal.HourlySalary;
    cout << "Enter the number of hours for each day\n";
    Mon = ObtainDailyHours("Monday");
    Tue = ObtainDailyHours("Tuesday");
    Wed = ObtainDailyHours("Wednesday");
    Thu = ObtainDailyHours("Thursday");
    Fri = ObtainDailyHours("Friday");
    Sat = ObtainDailyHours("Saturday");
    Sun = ObtainDailyHours("Sunday");

    Total = Mon + Tue + Wed + Thu + Fri + Sat + Sun;
    Seasonal.TotalHours = Total;
    Hourly = Seasonal.HourlySalary;
    Seasonal.WeeklySalary = CalcGrossPay(Total, Hourly);

    cout << "\nInformation about the employee";
    cout << "\n\tFull Name: "
         << Seasonal.FirstName << " " << Seasonal.LastName;
    cout << "\n\tTotal Weekly Hours: "
         << setiosflags(ios::fixed) << setprecision(2)
         << Seasonal.TotalHours;
    cout << "\n\tHourly Salary: $" << Seasonal.HourlySalary;
    cout << "\n\tWeekly Salary: $" << Seasonal.WeeklySalary;

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

15. Test the program twice to make sure it behaves appropriately. Test it once with less than 40 hours; test it again with overtime.

16. Return to Bcb

Techniques of Initializing a Class

There are various techniques used to initialize a class: initializing individual members or initializing the class as a whole.

To initialize a member of a class, access it and assign it an appropriate value.

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

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

#pragma argsused
class TShoeBox
{
public:
    Double Length, Width, Height;
    PChar Color;
    Single ShoeSize;
private:
};
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TShoeBox Shake;
    Double Volume;

    // Initializing each member of the class
    Shake.Length = 12.55;
    Shake.Width  = 6.32;
    Shake.Height = 8.74;
    Shake.Color  = "Yellow Stone";
    Shake.ShoeSize = 10.50;

    Volume = Shake.Length * Shake.Width * Shake.Height;

    // Display the characteristics of the shoe box
    cout << "Characteristics of this shoe box";
    cout << "\n\tLength = " << Shake.Length
         << "\n\tWidth = " << Shake.Width
         << "\n\tHeight = " << Shake.Height
         << "\n\tVolume = " << Volume
         << "\n\tColor = " << Shake.Color
         << "\n\tSize = " << Shake.ShoeSize;

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

This time, the program would render a reasonable result:

```
Characteristics of this shoe box
        Length = 12.55
        Width = 6.32
        Height = 8.74
        Volume = 693.222
        Color = Yellow Stone
        Size = 10.5

Press any key to continue...
```

You can also initialize an object as a variable. This time, type the name of the variable followed by the assignment operator, followed by the desired values of the variables listed between an opening and a closing curly brackets; each value is separated with a comma. The first rule you must keep in mind is that the list of variables must follow the order of the declared members of the class. The second rule you must observe is that none of the members of the class must be another class:

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

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

#pragma argsused
class TShoeBox
{
public:
    Double Length, Width, Height;
    PChar Color;
    Single ShoeSize;
private:
};
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    // Declaring and initializing the class as a variable
    TShoeBox LadyShake = { 12.55, 6.32, 8.74, "Yellow Stone", 10.50 };
    Double Volume = LadyShake.Length * LadyShake.Width * LadyShake.Height;

    // Display the characteristics of the shoe box
    cout << "Characteristics of this shoe box";
    cout << "\n\tLength = " << LadyShake.Length
         << "\n\tWidth = " << LadyShake.Width
         << "\n\tHeight = " << LadyShake.Height
         << "\n\tVolume = " << Volume
         << "\n\tColor = " << LadyShake.Color
         << "\n\tSize = " << LadyShake.ShoeSize;

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

Classes and Methods

The primary motivation of using classes in a program is to create objects as complete as possible. An object must be able to handle its own business so that the other objects of the program or of another program would only need to know which object can take care of a particular need they have.

A regular variable, as a member of an object, cannot handle assignments; this job is handled by particular functions declared as members of a class. A function as a member of a class is also called a Method. Therefore, in this book, the word “method”, when associate with a class, refers to a function that is a member of that class.

Declaring Methods

As a member of an object, a method is declared like any of the functions we have used so far; it could or could not return a value.

The shoe box we have been using so far needs a volume that would be used to determine what size can fit in the box. Therefore, we will use a member function

that can perform that calculation. Our object, when including methods could be structured as follows:

```
class TShoeBox
{
public:
    Double Length;
    Double Width;
    Double Height;
    PChar Color;
    Single ObtainShoeSize();
    Double CalcVolume();
private:
};
```

When using methods on a class, the variables are used to hold or store values, called data, of the object, while methods are used to perform assignments as related to the objects. One way you can control the data held by variables is to hide data from the "external world". To achieve this, you should declare the member variables in the private section. After doing this, use the methods in the public section to help the class interact with the other objects or functions of the program. At this time, our TShoeBox object would look like this:

```
class TShoeBox
{
public:
    Single ObtainShoeSize();
    Double CalcVolume();
    PChar Color;
private:
    Double Length;
    Double Width;
    Double Height;
};
```

There are at least two techniques you can use to implement a method member

Adding Methods to an Object

- To add methods to our project, change the content of the file as follows:

```
//---------------------------------------------------------------------------
class TEmployee
{
public:
    void IdentifyEmployee();
    void GetHourlySalary();
    void CalcTotalHours();
    void CalcGrossPay();
    void Display();
private:
    string FirstName;
    string LastName;
    double TotalHours;
    double HourlySalary;
    double GrossPay;
};
//---------------------------------------------------------------------------
```

Implementing Methods Locally

To implement a method in the class where it is declared, use the same techniques we used to define regular functions. When a method is a class' member, it has access to the member variables of the same class; this means you do not need to pass the variables as arguments (there are cases when you will need to); you can

just use any of them as if it were supplied. Here is how you would define the CalcVolume() method inside of the TShoeBox class:

```
class TShoeBox
{
public:
    Single ObtainShoeSize();
    Double CalcVolume()
    {
        return (Length * Width * Height);
    }
    PChar Color;
private:
    Double Length;
    Double Width Double Height;
};
```

If your class has a lot of methods, this technique could be cumbersome. You should use it only for small methods.

Implementing Methods Locally

1. To implement methods locally, change the content of the class as follows:

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

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

#pragma argsused
double __fastcall ObtainDailyHours(string s);
//---------------------------------------------------------------------------
class TEmployee
{
public:


    void IdentifyEmployee()
    {
        cout << "Enter the following pieces of information "
             << "about this employee\n";
        cout << "First Name: ";
        cin >> FirstName;
        cout << "Last Name: ";
        cin >> LastName;
    }

    void GetHourlySalary()
    {
        cout << "Hourly Salary: $";
        cin >> HourlySalary;
    }

    void CalcTotalHours()
    {
        double Mon, Tue, Wed, Thu, Fri, Sat, Sun;

        cout << "Enter the number of hours for each day\n";
        Mon = ObtainDailyHours("Monday");
        Tue = ObtainDailyHours("Tuesday");
        Wed = ObtainDailyHours("Wednesday");
        Thu = ObtainDailyHours("Thursday");
        Fri = ObtainDailyHours("Friday");
        Sat = ObtainDailyHours("Saturday");
        Sun = ObtainDailyHours("Sunday");
```

```
        // Calculate the total weekly hours
        TotalHours = Mon + Tue + Wed + Thu + Fri + Sat + Sun;
    }

    void CalcGrossPay()
    {
        if( TotalHours > 40 )
        {
            double OvertimeHours, RegularSalary, OvertimeGross;

            // If the total of weekly hours is above 40
            // then the employee has overtime
            RegularSalary = HourlySalary * 40;

            // The overtime is anything above 40 hours
            OvertimeHours = TotalHours - 40;

            // Hourly overtime salary is calculated by adding half
            // of the salary to the regular hourly salary
            double OvertimeSalary = HourlySalary + (HourlySalary * 0.50);

            // To calculate the overtime salary, multiply the overtime
            // hours to the overtime salary
            OvertimeGross = OvertimeHours * OvertimeSalary;

            // Now we have the regular salary and the overtime wage
            // Simply add them to get the gross weekly earnings
            GrossPay = RegularSalary + OvertimeGross;
        }
        // Since the employee's total hours are less than 40,
        // there is no overtime
        else
            GrossPay = HourlySalary * TotalHours;
    }

    void Display()
    {
        cout << "\nEmployee Payroll Information";
        cout << "\n\tFull Name: " << FirstName << " " << LastName;
        cout << "\n\tTotal Weekly Hours: "
             << setiosflags(ios::fixed) << setprecision(2)
             << TotalHours;
        cout << "\n\tHourly Salary: $" << HourlySalary;
        cout << "\n\tWeekly Salary: $" << GrossPay;
    }

private:
    string FirstName;
    string LastName;
    double TotalHours;
    double HourlySalary;
    double WeeklySalary;
    double GrossPay;
};
//---------------------------------------------------------------------------
double __fastcall ObtainDailyHours(string s)
{
    double h;

    do {
        cout << s << ": ";
        cin >> h;
        if(h < 0 || h > 24)
            cout << "Please enter a number between 0.00 and 24.00\n";
    }while(h < 0 || h > 24);

    return h;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
```



```
    TEmployee Contractor;

    Contractor.IdentifyEmployee();
    Contractor.GetHourlySalary();
    Contractor.CalcTotalHours();
    Contractor.CalcGrossPay();
    Contractor.Display();

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

2. To test the program, on the main menu, click Run ª Run. Here is an example of running the program:

```
Enter the following pieces of information about this employee
First Name: Jeanine
Last Name: Bliss
Hourly Salary: $14.55
Enter the number of hours for each day
Monday: 8
Tuesday: 9.50
Wednesday: 7.50
Thursday: 8.50
Friday: 8
Saturday: 0
Sunday: 0

Employee Payroll Information
        Full Name: Jeanine Bliss
        Total Weekly Hours: 41.50
        Hourly Salary: $14.55
        Weekly Salary: $614.74

Press any key to continue...
```

3. Return to Bcb

Implementing Methods Globally

When the methods execute long assignments, you should implement them outside of the object by first accessing the desired function member of the class. To access a method of a class when implementing it, instead of the member access operator ".", you will use the scope resolution operator represented as two colons ::

To implement a method outside of the class, type the return value of the method, followed by the class' name, followed by the scope resolution operator "::", followed by the method's name, followed by the arguments, if any, between parentheses, and finally define what the function should do, in its body.

Another implementation of our CalcVolume() method would be:

```
//---------------------------------------------------------------------------
class TShoeBox
{
public:
    Single ObtainShoeSize();
    Double CalcVolume();
    PChar Color;
private:
    Double Length;
    Double Width;
    Double Height;

};
```

```
//---------------------------------------------------------------------------
Double TShoeBox::CalcVolume()
{
    return (Length * Width * Height);
}
//---------------------------------------------------------------------------
```

Implementing Methods Globally

1. To implement the methods globally, change the content of the file as follows:

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

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

#pragma argsused
double __fastcall ObtainDailyHours(string s);
//---------------------------------------------------------------------------
class TEmployee
{
public:
    void IdentifyEmployee();
    void GetHourlySalary();
    void CalcTotalHours();
    void CalcGrossPay();
    void Display();

private:
    string FirstName;
    string LastName;
    double TotalHours;
    double HourlySalary;
    double WeeklySalary;
    double GrossPay;
};
//---------------------------------------------------------------------------
void TEmployee::IdentifyEmployee()
{
    cout << "Enter the following pieces of information "
         << "about this employee\n";
    cout << "First Name: ";
    cin >> FirstName;
    cout << "Last Name: ";
    cin >> LastName;
}
//---------------------------------------------------------------------------
void TEmployee::GetHourlySalary()
{
    cout << "Hourly Salary: $";
    cin >> HourlySalary;
}
//---------------------------------------------------------------------------
void TEmployee::CalcTotalHours()
{
    double Mon, Tue, Wed, Thu, Fri, Sat, Sun;

    cout << "Enter the number of hours for each day\n";
    Mon = ObtainDailyHours("Monday");
    Tue = ObtainDailyHours("Tuesday");
    Wed = ObtainDailyHours("Wednesday");
    Thu = ObtainDailyHours("Thursday");
    Fri = ObtainDailyHours("Friday");
    Sat = ObtainDailyHours("Saturday");
    Sun = ObtainDailyHours("Sunday");

    // Calculate the total weekly hours
```



```
    TotalHours = Mon + Tue + Wed + Thu + Fri + Sat + Sun;
}
//---------------------------------------------------------------------------
void TEmployee::CalcGrossPay()
{
    if( TotalHours > 40 )
    {
        double OvertimeHours, RegularSalary, OvertimeGross;

        // If the total of weekly hours is above 40
        // then the employee has overtime
        RegularSalary = HourlySalary * 40;

        // The overtime is anything above 40 hours
        OvertimeHours = TotalHours - 40;

        // Hourly overtime salary is calculated by adding half
        // of the salary to the regular hourly salary
        double OvertimeSalary = HourlySalary + (HourlySalary * 0.50);

        // To calculate the overtime salary, multiply the overtime
        // hours to the overtime salary
        OvertimeGross = OvertimeHours * OvertimeSalary;

        // Now we have the regular salary and the overtime wage
        // Simply add them to get the gross weekly earnings
        GrossPay = RegularSalary + OvertimeGross;
    }
    // Since the employee's total hours are less than 40,
    // there is no overtime
    else
        GrossPay = HourlySalary * TotalHours;
}
//---------------------------------------------------------------------------
void TEmployee::Display()
{
    cout << "\nEmployee Payroll Information";
    cout << "\n\tFull Name: " << FirstName << " " << LastName;
    cout << "\n\tTotal Weekly Hours: "
         << setiosflags(ios::fixed) << setprecision(2)
         << TotalHours;
    cout << "\n\tHourly Salary: $" << HourlySalary;
    cout << "\n\tWeekly Salary: $" << GrossPay;
}
//---------------------------------------------------------------------------
double __fastcall ObtainDailyHours(string s)
{
    double h;

    do {
        cout << s << ": ";
        cin >> h;
        if(h < 0 || h > 24)
            cout << "Please enter a number between 0.00 and 24.00\n";
    }while(h < 0 || h > 24);

    return h;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TEmployee Contractor;

    Contractor.IdentifyEmployee();
    Contractor.GetHourlySalary();
    Contractor.CalcTotalHours();
    Contractor.CalcGrossPay();
    Contractor.Display();

    cout << "\n\nPress any key to continue...";
    getchar();
```

```
        return 0;
    }
    //---------------------------------------------------------------------------
```

2. To test the program, press F9.
3. Return to Bcb

Inline Methods

When studying functions, we learned that an assignment can be carried where it is being called. The same process can apply to a class' member.

To declare a class' method as inline, precede its name with the inline keyword when declaring the method in the class:

```
//---------------------------------------------------------------------------
class TShoeBox
{
public:
    inline Double CalcVolume();
    Single CalcShoeSize();
    void Display();
private:
    . . .
};
//---------------------------------------------------------------------------
```

You can choose which methods would be inline and which ones would not. When implementing the method, you can precede the method with the inline keyword. You can also omit the inline keyword in the class but use it when defining the method.

If you decide to implement a method locally (in the class), you have the option of implementing it as inline:

```
//---------------------------------------------------------------------------
class TShoeBox
{
public:
    inline Double CalcVolume()
    {
        return Length * Width * Height;
    }
    inline Single CalcShoeSize()
    {
        return Length - 0.35;
    }
    void Display();

private:
    Double Length;
    Double Width;
    Double Height;
    PChar Color;
};
//---------------------------------------------------------------------------
```

On the other hand, if you omit the inline keyword, the C++ Builder compiler would take care of it. Normally, any function implemented in the body of the class is considered inline.

Using Inline Methods

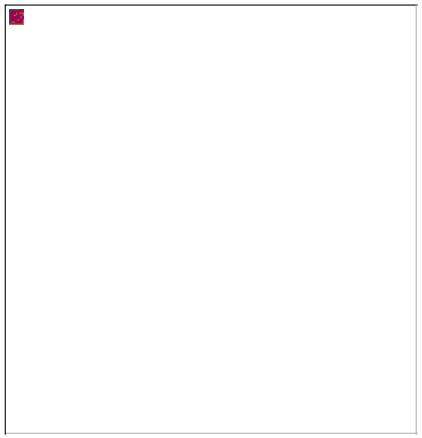
1. To declare a new variable, expand the project name in the Class Explorer and expand the TEmployee class folder. Right-click TEmployee (in the Class Explorer) and click New Field...

2. In the Field Name edit box, type NetPay
3. In the Type combo box, type d and double-click the down-pointing arrow. The keyword double will be selected.
4. In the Visibility section, click the Private radio button:

5. Click OK
6. To declare an inline method, right-click TEmployee in the Class Explorer and click New Method...:

7. In the Method Name edit box, type IsMarried

8. Click the arrow of the Function Result combo box and select bool
9. In the Visibility section, click the Public radio button.
10. In the Implementation Details section, click the Inline check box



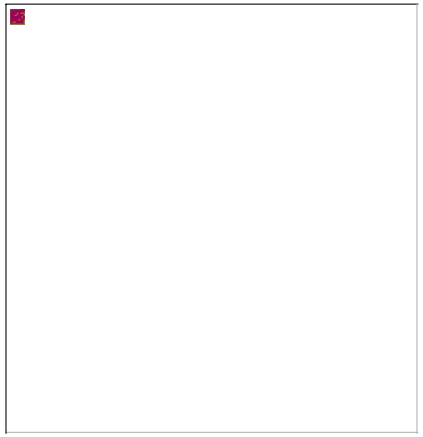
11. Click OK. The function will be added to the end of the file. As long as the scope resolution links the function to the class, the position of the function on the file is not important.
12. Implement the method as follows:

```
//---------------------------------------------------------------------------
bool inline TEmployee::IsMarried()
{
    char Answer;

    cout << "Are you married(y=Yes/n=No)? ";
    cin >> Answer;

    if( Answer == 'y' || Answer == 'Y' )
        return true;
    else
        return false;
}
//---------------------------------------------------------------------------
```

13. To create an implicit inline method, right-click the TEmployee class in the Class Explorer and click New Method.
14. Set the name of the method as CalcNetPay
15. Set the result type as void
16. In the Implementation Details section, click both the Inline and the Implicit Inline check boxes:



17. Click OK.
18. Notice that the method's declaration and its body are ready in the TEmployee class.
19. Implement the method as follows:

```
//---------------------------------------------------------------------------
class TEmployee
{
public:
    ...

    void CalcNetPay()
    {
        if( IsMarried() == false )
            NetPay = CalcGrossPay() - (CalcGrossPay() * 30 / 100);
        else
            NetPay = CalcGrossPay();
    }

private:
    ...
};
//---------------------------------------------------------------------------
```

20. Change the Display() method and the main() function as follows:

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

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

#pragma argsused
double __fastcall ObtainDailyHours(string s);
//---------------------------------------------------------------------------
class TEmployee
{
...

private:
    ...
};
```

```
//---------------------------------------------------------------------------
...
//---------------------------------------------------------------------------
void TEmployee::Display()
{
    cout << "Employee Payroll";
    cout << "\nFull Name: " << FirstName << " " << LastName;
    cout << "\nTotal Weekly Hours: "
         << setiosflags(ios::fixed) << setprecision(2)
         << TotalHours;
    cout << "\nHourly Salary: $" << HourlySalary;
    cout << "\nWeekly Salary: $" << GrossPay;
    cout << "\nNet Weekly Salary: $" << NetPay;
}
//---------------------------------------------------------------------------
double __fastcall ObtainDailyHours(string s)
{
    double h;

    do {
        cout << s << ": ";
        cin >> h;
        if(h < 0 || h > 24)
            cout << "Please enter a number between 0.00 and 24.00\n";
    }while(h < 0 || h > 24);

    return h;
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TEmployee Contractor;

    Contractor.IdentifyEmployee();
    Contractor.GetHourlySalary();
    Contractor.CalcTotalHours();
    Contractor.CalcGrossPay();
    Contractor.CalcNetPay();
    clrscr();
    Contractor.Display();


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

21. To test the program, press F9.
22. Return to Bcb

Class Members Interactions

Regardless of how the member methods of an object are implemented, any method can call another without using an access operator. This allows an object's methods to exchange information among themselves easily. Furthermore, unlike regular functions where a function must be declared prior to another function calling it, the method members of a class do not abide by that rule: one method can call another method before or after the other has been implemented, as long as it is defined somewhere.

```
//---------------------------------------------------------------------------
class TShoeBox
{
public:
    . . .
};
//---------------------------------------------------------------------------
```



```
void TShoeBox::Display()
{
    // Initializing the dimensions
    Length = 12.55;
    Width = 6.32;
    Height = 8.74;
    Color = "Early Breeze";

    // Display the characteristics of the shoe box
    cout << "Characteristics of this shoe box";
    cout << "\n\tLength = " << Length
         << "\n\tWidth = " << Width
         << "\n\tHeight = " << Height
         << "\n\tVolume = " << CalcVolume()
         << "\n\tColor = " << Color
         << "\n\tSize = " << CalcShoeSize();
}
//---------------------------------------------------------------------------
Double TShoeBox::CalcVolume()
{
    return Length * Width * Height;
}
//---------------------------------------------------------------------------
```

Once an object is defined and behaves as complete as possible, the other function or objects of the program can make the appropriate calls trusting that the called object can handle its assignments efficiently. This ability allows you to (tremendously) reduce the work overload of the other components of a program.

The main() function can simply call the appropriate member of the TShoeBox object now:

```
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TShoeBox Sample;
    Sample.Display();

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

Object Methods Interactions

1. To let a method call other member methods of the same object, change the Display() method and the main() function as follows:

```
//---------------------------------------------------------------------------
void TEmployee::Display()
{
    // Execute the prerequisite methods to prepare a payroll
    IdentifyEmployee();
    GetHourlySalary();
    CalcTotalHours();
    CalcGrossPay();
    CalcNetPay();
    clrscr();

    cout << "Employee Payroll";
    cout << "\nFull Name: " << FirstName << " " << LastName;
    cout << "\nTotal Weekly Hours: "
         << setiosflags(ios::fixed) << setprecision(2)
         << TotalHours;
    cout << "\nHourly Salary: $" << HourlySalary;
    cout << "\nWeekly Salary: $" << GrossPay;
    cout << "\nNet Weekly Salary: $" << NetPay;
```

```
}
//---------------------------------------------------------------------------
double __fastcall ObtainDailyHours(string s)
{
    . . .
}
//---------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    TEmployee Contractor;

    Contractor.Display();

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

2. To test the program, press F9.
3. After running the program, return to Bcb

Previous

Next