



# Variables and Data Types

To perform its assignments, you use items of various kinds. For example, to calculate the area of a circle, you would use a radius. To identify an employee, you could use a name. To count the books in a bookstore, you use a number. These items are used to recognize the components of a program. It is very usual that the values given to these items change regularly throughout the lifetime of a program. For example, if you write a program used to register orders for a car repair shop, two customers will hardly have the same issue. Starting with customers names, you will likely deal with different names throughout the day. Since you will not write a new program for each customer, you would make sure that one program can process every kind of order. Such a program would have an item used to get customers' names. Since customers' names are different, they vary from one order to another. Therefore, an item such as the one taking a customer's name, since it can change (vary) from one customer to another, is called a variable.

To make the compiler's job easier, every one of these items or variable must have a name. Once you provide a name, the compiler can use the same item throughout a program.

## C++' Names

When using the various necessary variables in your programs, you will need to identify each one of them. A variable is primarily recognized by its name. C++ provides rules for naming items in your program.

The name of a variable:

- Starts with an underscore " \_ " or a letter, lowercase or uppercase, such as a letter from a to z or from A to Z. Examples are Name, gender, \_Students, pRice
- Can include letters, underscore, or digits. Examples are: keyboard, Master, Junction, Player1, total\_grade, \_Score\_Side1
- Cannot include special characters such as !, %, ], or \$
- Cannot include an empty space
- Cannot be any of the reserved words
- Should not be longer than 32 characters (although allowed)

A name can consist of one word such as country. A name could also be a combination of more than one word, such as firstname or dateofbirth.

The C++ compiler has a list of words reserved for its own use and you must not use any of these words to name your own objects or functions. The reserved words are:

auto	do	goto	signed	union
break	double	if	sizeof	unsigned
case	else	int	static	void
char	enum	long	struct	volatile
const	extern	register	switch	while
continue	float	return	typedef	
default	for	short		

Some of these words are used by Borland C++ Builder or could come in conflict

with the libraries used in Microsoft Windows. Therefore, avoid using these additional words:

asm	dynamic_cast	namespace	reinterpret_cast	try
bool	explicit	new	static_cast	typeid
catch	false	operator	string	typename
class	friend	private	template	union
cin	inline	protected	this	using
const_cast	interrupt	public	throw	virtual
cout	mutable	register	true	wchar_t
delete				

Here are other words you should avoid using when programming in Borland C++ Builder:

__export	__near	__huge	ada	PASCAL
__fastcall	__pascal	__import	AnsiString	False
__huge	__rtti	__interrupt	entry	FALSE
__import	__saveregs	__loadds	far	True
__int16	__seg	__near	fortran	TRUE
__int32	__ss	__pascal	huge	
__int64	__stdcall	__saveregs	near	
__int8	__thread	__seg	pascal	
__interrupt	__try	__ss		
__loads		__stdcall		

Avoid starting the name of a variable with two underscores; sometimes, the compiler would think that you are trying to use one of the words reserved for the compiler.

C++ is case-sensitive; this means CASE, Case, case, and Case are four completely different words. To make your programming experience easier and personal, you can add your own rules to those above. Some (most) companies also adopt a naming convention throughout their documentation.

Throughout this book, a name will:

- Start in uppercase, whether it is a variable, a function, a structure, or a class. In some situations, a name will start with an underscore, when useful. Examples are Country, Printer, \_Number
- Start each element of its combined names in uppercase. Examples are FirstName, DateOfBirth

The preprocessor is used to give a special instruction to the compiler. It is typically placed at the beginning of a line and it is followed by a word that would perform an action. The formula to use the preprocessor is:

#action What-To-Do

There are various actions you could ask the compiler to perform.

## Preprocessors: #include

The #include is used to include an external file in the current file. The file to include is called a header file, a library, or another kind of file you want the compiler to consider before going any further. For example, to include one of the header files that shipped with Bcb, such as calendar.h, you can write

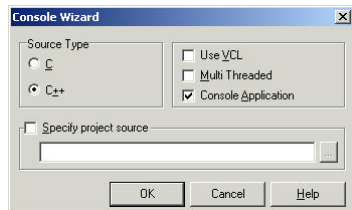
```
#include <calendar.h>
```

To include one of your header files when you will have created one, you can write

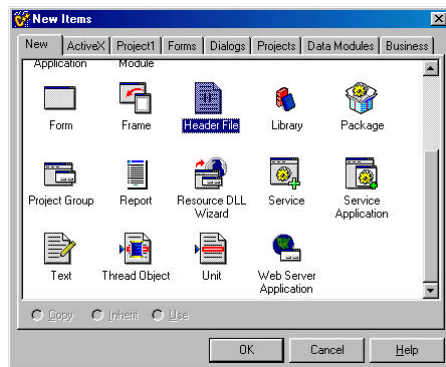
```
#include "Circle.h"
```

### ◆ Using the #include Preprocessor

1. Start Borland C++ Builder
2. On the main menu, click File -> New... or File -> New -> Other...
3. In the New Items dialog box, make sure you are in the New property sheet. Click Console Wizard and click OK.
4. In the Console Wizard dialog box, make sure the C++ radio button is selected and the Console Application check box is checked



5. Click OK.
6. Delete the content of the whole file.
7. On the main menu, click File -> New...
8. In the New Items dialog, click Header File:



9. Click OK.
10. In the File1.h empty file, type:

```
#include <iostream.h>
```

11. Click the Unit1.cpp tab.
12. In the empty file, type

```
#include "File1.h"
```

## Preprocessors: #define

The #define, called a directive, is used to direct the compiler to create or perform a (small) action. This action is called a macro. For example, you can ask the compiler to use "United States of America" whenever it sees USA. To do that you can write

```
#define USA "United States of America"
```

If you insert the word USA anywhere in your program, such as cout << USA, the compiler would replace it with the defined name. You can also use the #define directive to create words that would be replaced with numeric values. In fact, you can use the #define directive to create a small/mini interpreter.

### ◆ Using the #define Preprocessor

1. Click the File1.h tab and add the following lines to it:

```
#include <iostream.h>
#define main main()
#define Begin {
#define End }
#define USA "United States of America"
#define WH "The White House"
#define Address "1600 Pennsylvania Avenue"
#define Eof "Press any key to continue..."
#define GT getchar()
#define Show cout <<
#define Eol << '\n'
```

2. Click the Unit1.cpp tab and modify it with the following:

```
#include "File1.h"
main
Begin
    Show USA Eol;
    Show WH Eol;
    Show Address Eol;
    Show Eof;
    GT;
End
```

3. To test the program, press F9

## Preprocessors: #pragma

Every compiler in the industry has characteristics that distinguish it from other compilers. When building or compiling a program, you should be sensitive to the environment the program would be running. The #pragma preprocessor is used to give specific instructions to the compiler based on the particular compiler that is being used to create the program. This means the #pragma directive completely depends on the compiler you are using.

- At the risk of repeating the provided documentation (and for copyright issues), please consult Borland C++ Builder (and Microsoft Visual C++) help files. These two companies spent a lot of time documenting the preprocessors. The #pragma preprocessor is highly documented on both compilers. Do a search on the word pragma.

The #pragma hdrstop ( for hdrstop, pronounce "Header Stop") provides a line or block separation between two sections. When using header, include the headers that came with Bcb before this preprocessor, like this:

```
#include <iostream.h>
#include <conio.h>
#include <vcl.h>
#pragma hdrstop
```

Include the headers that were created during the development of your program. Our file could then be:

```
#include <iostream.h>
#include <conio.h>
#include <vcl.h>
#pragma hdrstop

#include "File1.h"
#include "Circle.h"
```

### ◆ Using the #pragma Preprocessor

1. Click the Unit1.cpp tab. Change its content as follows:

```
#pragma hdrstop
#include "File1.h"
main
Begin
    Show USA Eol;
    Show WH Eol;
    Show Address Eol;
    Show Eof;
    GT;
End
```

2. Click the File1.h tab and modify it as follows:

```
#include <iostream.h>
#include <conio.h>
#include <vcl.h>
#pragma hdrstop
#include <math.h>
#define main main()
#define Begin {
#define End }
#define USA "United States of America"
#define WH "The White House"
#define Address "1600 Pennsylvania Avenue"
#define Eof "Press any key to continue..."
#define GT getchar()
#define Show cout <<
#define Eol << '\n'
```

3. To test the program, press F9

## Introduction to Variables

When you are planning to use a variable whose value is not determined in advance, you should let the compiler know. The compiler will reserve a space in memory for that variable.

Variable1

That space is still empty and has no value in it. Initialization is a technique of assigning a set value to a variable. The compiler takes a value and "puts" it in the reserved memory space. There are two main techniques used to initialize a variable.

To use the assignment operator to initialize a variable, write:

Variable = InitialValue;

Another technique of initializing a variable is by using parentheses. The syntax is:

Variable(InitialValue);

## The cout Extractor Operator Revisited

There are two main uses you will make of an item in your program: to display its value on the screen or to get its value from the user. To display an item on the screen, you can use the cout operator. By default, the cout operator is used to display a sentence or an expression on one line. Such a sentence would be included in double-quotes. An example would be:

```
cout << "This is our C++ tutorial";
```

If you want to display something else, you can write another statement on the subsequent line(s). The cout operator can expand on more than one line. To do that, do not end the first line with a semi-colon. If you are displaying a string (a group of words between two double quotes), close the string with double-quotes even if you are in the middle of the string, then start the following line with a << operator followed by a double-quote and the next part of the string. Do not forget to close the string with a double quote. For example, you could display the following address using various cout statements:

```
cout << "The White House";
cout << "1600 Pennsylvania Avenue";
```

Or you could use one cout operator, like this:

```
cout << "The White House "
<< "1600 Pennsylvania Avenue";
```

The cout operator is also typically used to display values of variables or expressions on the screen. The value, represented by its variable, or an expression is not enclosed in double quotes. For example, to display an employee's name, you could write:

```
cout << EmployeeName;
```

The preceding line could result in displaying:

Albert

To make your program easier to understand you should always use a small introduction to let the user identify what is really displaying on the screen. To do that, you enclose your introduction as a string, between double-quotes, and then let the compiler display the appropriate item. Each section of your statement will be separated from the other with a double less than sign. For example, to display the above example, you could write:

```
cout << "This is the employee's name: " << employeename;
```

### ◆ Using the cout Operator

1. Start C++ Builder
2. On the main menu, click File -> New...
3. In the New Items dialog box, make sure you are in the New property page displays. Click Console Wizard and click OK.
4. In the Console Wizard dialog box, make sure the C++ radio button is selected and the Console Application check box is checked.
5. Click OK.

6. Change the content of the file as follows (you will delete the other words):

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    cout << "Student Age: ";

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

7. To test your program, press F9

8. To display a long statement using a single cout operator, change the content of the file as follows:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    cout << "Here is a piece of information about the student"
        << "\nStudent Age: ";

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

9. Execute your program to see the result

10. To display a value or an expression using the cout operator, change the content of the file as follows:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    cout << "Here is a piece of information about the student"
        << "\nStudent Age: ";
    cout << Age;

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

11. Execute the program.  
You will receive at least one error; and the cursor should be blinking on it. We will solve that error shortly:

[C++ Error] Unit1.cpp(12): E2451 Undefined symbol 'Age'

## The cin Operator

C++ is equipped with another operator used to request values from the user. The user usually provides such a value by typing it using the keyboard. The cin (pronounce "see in") operator is used for that purpose; it displays a blinking cursor on the monitor to let the user know that a value is expected. Unlike the cout operator, the cin uses two greater than signs ">>" followed by the name of the expected value. The syntax of the cin operator is:

```
cin >> ValueName;
```

If you are requesting many values from the user, you can write a statement that would act accordingly by separating values with as many >> operators as necessary. For example, to request a first name, last name, and an age on the same line, you could use:

```
cin >> FirstName >> LastName >> Age;
```

### ◆ Using the cin Operator

1. To request a value from the user, change the content of the file as follows:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    cout << "Enter the student's age: ";
    cin >> Age;
    cout << "You typed " << Age;

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

2. Execute the program. Once again, you receive an error.

## Variables and Data Types

The error we have received in the preceding programs means the compiler does not understand what the word Age means. This is because, before using any item in your program, the compiler wants to be aware of it. The compiler wants to identify it.

When you let the compiler know that you want to use a certain variable, it reserves an amount of space in memory appropriate for the object. Different objects use different amounts of space in memory and the compiler needs to know how much space you will need.

Letting the compiler know about a variable is referred to as declaring the variable. A variable is declared with two entities: its type and its name. The type of the object is also called a data type. The syntax of declaring a variable is:

```
DataType VariableName;
```

If many different variables are using the same data type, you can declare them on the same line, separating two with a comma, except the last one that would end with a semi-colon, as follows:

```
DataType Variable1, Variable2, Variable3;
```

## Boolean Variables

The Boolean data type is used to declare a variable whose value would be set as true (1) or false (0). To declare such a value, you use the bool, the Boolean, the BOOL, or the BOOLEAN keywords. The variable can then be initialized with a starting value. The Boolean constant is used to check that the state of a variable (or a function) is true or false. You can declare such a variable as:

```
bool GotThePassingGrade = true;
```

Later in the program, for a student who got a failing grade, you can assign the other value, like this

```
GotThePassingGrade = false;
```

The compiler we are using recognizes the words Boolean and BOOL as data types for Boolean variables. The values of the bool, Boolean, and BOOL variables can be true, True, TRUE, false, False, or FALSE. In this book, we will use the bool, Boolean, and BOOL identifiers interchangeably. We will also use true, True, and TRUE or false, False, and FALSE interchangeably.

### ♦ Using Boolean Variables

1. Change the content of the file as follows:

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

#pragma argsused
int main(int argc, char* argv[])
{
    bool MachineIsWorking = true;

    cout << "Since this machine is working, its value is "
         << MachineIsWorking << endl;

    MachineIsWorking = false;
    cout << "The machine has stopped operating. "
         << "Now its value is " << MachineIsWorking << endl;

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

2. Test the program and return to Bcb
3. To start a new project, on the Standard toolbar, click the New button
4. On the New Items dialog, double-click Console Wizard.
5. On the Console Wizard, click OK because Bcb remembers the last selections you made.
6. At the question of whether you want to save your project, click Yes.

7. Click the arrow of the Save In combo box and click My Documents.
8. Once My Documents displays in the Save In combo box, click the Create New Folder button
9. Type Bool and press Enter
10. Double-click the newly created folder to display it in the Save In combo box.
11. Click Save twice.

## Enumerations

An enumeration provides a technique of setting a list of numbers where each item of the list is ranked and has a specific name. Therefore, instead of numbers that represent players of a football (soccer) game such as 1, 2, 3, 4, 5, you can use names instead. This would produce GoalKeeper, RightDefender, LeftDefender, Stopper, Libero. The formula of creating an enumeration is

```
enum Series_Name {Item1, Item2, Item_n};
```

In our example, the list of players by their name or position would be

```
enum Players { GoalKeeper, RightDefender, LeftDefender, Stopper, Libero };
```

Each name in this list represents a constant number. Since the enumeration list is created in the beginning of the program, or at least before using any of the values in the list, each item in the list is assigned a constant number. The items are counted starting at 0, then 1, etc. By default, the first item in the list is assigned the number 0, the second is 1, etc. Just like in a game, you do not have to follow a strict set of numbers. Just like a goalkeeper could have number 2 and a midfielder number 22, you can assign the numbers you want to each item, or you can ask the list to start at a specific number.

In our list above, the goalkeeper would have No. 0. To make the list start at a specific number, assign the starting value to the first item in the list. Here is an example:

```
enum Players { GoalKeeper = 12, RightDefender, LeftDefender, Stopper, Libero };
```

This time, the goalkeeper would be No. 12, the RightDefender would be No. 13, etc.

You still can assign any value of your choice to any item in the list, or you can set different ranges of values to various items. You can create a list like this:

```
enum Days { Mon, Tue, Wed, Thu, Fri, Sat, Sun = 0 };
```

Since Sun is No. 0, Sat would be No. -1, Fri would be -2 and so on. On the other hand, you can set values to your liking. Here is an example:

```
enum Colors {Black = 2, Green = 4, Red = 3, Blue = 5, Gray, White = 0 };
```

In this case, the Gray color would be 6 because it follows Blue = 5.

Once the list has been created, the name you give to the list, such as Players, becomes an identifier of its own, and its can be used to declare a variable. Therefore, a variable of the enumerated type would be declared as:

```
Series_Name Variable;
```

You can declare more than one variable of such a type. An example of the type of our list of players would be:

Players Defense, Midfield, Attack;  
 Players HandBall, BasketBall, VolleyBall;

Instead of declaring variables after the list has been created, you can declare the variables of that type on the right side of the list but before the closing semi-colon. Here is an example of a color enumeration and variables declared of that type:

```
enum Flags {Yellow, Red, Blue, Black, Green, White} Title, Heading1, BottomLine;
```

To name the enumerators, Borland uses a convention. The name of the enumerator starts with T. If the name represents one word, the first letter of the word would be in uppercase. For example, to create an enumerator that represents hats, the enumerator would be named THats. If the name is a combination of different words, the starting letter of each is in uppercase. For example, an enumerator that represents hat categories would be called THatCategories. To name the elements that compose the enumerator, each name start with two lower case letters. The first letter of each element's name would be in uppercase. If the name is a combination of words, each first letter would be in uppercase. For example, to create an enumerator that represent book categories, the name would be

```
TBookCategories { bcBiographies, bcComputerSciences, bcHistory };
```

We will follow the same convention here.

## ◆ Using Enumerators

1. Change the content of the file as follows:

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

#pragma argsused
int main(int argc, char* argv[])
{
    enum TPizzaSize {psSmall, psMedium, psLarge, psJumbo};

    TPizzaSize Size;
    cout << "The small pizza has a value of " << psSmall << endl;
    cout << "The medium pizza has a value of " << psMedium << endl;
    cout << "The large pizza has a value of " << psLarge << endl;
    cout << "The jumbo pizza has a value of " << psJumbo << endl;

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

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

```
The small pizza has a value of 0
The medium pizza has a value of 1
The large pizza has a value of 2
The jumbo pizza has a value of 3

Press any key to continue...
```

3. To assign your own values to the list, change the enumeration as follows:

```
enum TPizzaSize {psSmall = 4, psMedium = 10, psLarge = 16, psJumbo = 24};
```

4. To test the program, on the Standard toolbar, click the Run button.

5. To get the starting as you want, change the enumeration as follows:

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

#pragma argsused
int main(int argc, char* argv[])
{
    enum TPizzaSize {psSmall = 4, psMedium, psLarge, psJumbo};

    TPizzaSize Size;
    cout << "The small pizza has a value of " << psSmall << endl;
    cout << "The medium pizza has a value of " << psMedium << endl;
    cout << "The large pizza has a value of " << psLarge << endl;
    cout << "The jumbo pizza has a value of " << psJumbo << endl;

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

6. To test the program, press F9

```
The small pizza has a value of 4
The medium pizza has a value of 5
The large pizza has a value of 6
The jumbo pizza has a value of 7

Press any key to continue...
```

7. To assign values at random, change the list of pizzas as follows:

```
enum TPizzaSize {psSmall = 2, psMedium, psLarge = 14, psJumbo = -5};
```

8. Test the program

```
The small pizza has a value of 2
The medium pizza has a value of 3
The large pizza has a value of 14
The jumbo pizza has a value of -5

Press any key to continue...
```

9. To see another variant of declaring variables of an enumeration type, change the list line as follows:

```
enum TPizzaSize {psSmall, psMedium, psLarge, psJumbo} Full, Slide, Approach;
```

10. Test and the program and return to Bcb.

11. Another technique used to create an enumerator that has many members consists of writing each member on its own line. To see an example, change the declaration of the enumerator as follows:

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

#pragma argsused
int main(int argc, char* argv[])
{
    enum TPizzaSize {
```

```

    psSmall = 120,
    psMedium = -2666,
    psLarge = -404,
    psJumbo = 58
};

cout << "The small pizza has a value of " << psSmall << endl;
cout << "The medium pizza has a value of " << psMedium << endl;
cout << "The large pizza has a value of " << psLarge << endl;
cout << "The jumbo pizza has a value of " << psJumbo << endl;

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

```

12. Test the program and return to Bcb.

13. You can also use this technique to declare enumerator variables. As an example, change the enumerator as follows:

```

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

#pragma argsused
int main(int argc, char* argv[])
{
    enum TPizzaSize {
        psSmall = 120,
        psMedium = -2666,
        psLarge = -404,
        psJumbo = 58
    } Full, Slide, Approach;

    cout << "The small pizza has a value of " << psSmall << endl;
    cout << "The medium pizza has a value of " << psMedium << endl;
    cout << "The large pizza has a value of " << psLarge << endl;
    cout << "The jumbo pizza has a value of " << psJumbo << endl;

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

```

14. Test the program and return to Bcb.

## Integers

In C++, an integer is a natural number typically used to count items.

int, Integer, INT

To declare a variable as an integer, use the int, the Integer, or the INT keywords. The integer data type is used for a variable whose value can range from -2147483648 to 2147484647.

The following program declares all three variants of the integer data type:

```

//-----
#include <vc1.h>
#include <iostream.h>
#pragma hdrstop

```

```

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    int CoordX;
    Integer CoordY;
    INT CoordZ;

    cout << "Enter the coordinates of point A\n";
    cout << "Horizontal X = ";
    cin >> CoordX;
    cout << "Vertical Y = ";
    cin >> CoordY;
    cout << "Depth Z = ";
    cin >> CoordZ;

    cout << "\nOn a cartesian system, point A is located at";
    cout << "\n\tX = " << CoordX;
    cout << "\n\tY = " << CoordY;
    cout << "\n\tZ = " << CoordZ;

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

```

Compiled, the program would produce:

```

Enter the coordinates of point A
Horizontal X = -12
Vertical Y = 8
Depth Z = 6

```

```

On a cartesian system, point A is located at
    X = -12
    Y = 8
    Z = 6
Press any key to continue...

```

signed

The signed is used to designate an integer variable whose value could be negative or positive. You can also use the INT where you would use the signed keyword.

short, SHORT, and Smallint

The short keyword, its redefined type Smallint, and the Win32 SHORT are used to identify numeric variables on a 16-bit operating system. They are integers that typically range from -32768 to 32767.

```

//-----
#include <vc1.h>
#include <iostream.h>
#pragma hdrstop
//-----

#pragma argsused
int main(int argc, char* argv[])
{
    short    Number1;
    Smallint Number2;
    INT      Number3;

    cout << "Enter a number between -32768 and 32767: ";
    cin >> Number1;
    cout << "Enter another number: ";

```

```

cin >> Number2;
cout << "Enter at third number: ";
cin >> Number3;

cout << "\nThe numbers you entered were";
cout << "\n\tNumber 1: " << Number1;
cout << "\n\tNumber 2: " << Number2;
cout << "\n\tNumber 3: " << Number3;

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

```

This program would produce:

```

Enter a number between -32768 and 32767: 4288
Enter another number: -125
Enter at third number: 60

```

```

The numbers you entered were
Number 1: 4288
Number 2: -125
Number 3: 60
Press any key to continue...

```

long, signed long, Longint, and LONG

The long and LONG keywords are used to identify a 32-bit integer. The long, the signed long, or the LONG apply to a numeric value that could be either negative or positive.

unsigned

The unsigned keyword is used to designate an integer variable whose value must always be positive. (Most of the time, the unsigned is appended to an int, a short, a long, or a char to qualify them as positive.)

```

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    unsigned DayOfBirth, MonthOfBirth, YearOfBirth;

    cout << "The following pieces of information "
         << "are required for each student\n";
    cout << "Day of Birth: ";
    cin >> DayOfBirth;
    cout << "Month of Birth: ";
    cin >> MonthOfBirth;
    cout << "Year of Birth: ";
    cin >> YearOfBirth;
    cout << "\nStudent's Date of Birth: "
         << DayOfBirth << "/" << MonthOfBirth << "/" << YearOfBirth;

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

```

unsigned short, Word, WordBool, USHORT, and WORD

The unsigned short, its redefined Borland equivalents Word and WordBool, the Win32 USHORT and WORD are used to identify a 16-bit positive integer that would range from 0 to 65535. Borland C++ Builder provides a good mechanism of checking the validity of a Word, USHORT, a WORD, and usually an unsigned short variable; it would make sure that the provided value is positive. If the value is not positive, instead of raising an error (called an exception), the compiler would reset the value to 0.

```

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    unsigned short  Shirts;
    Word            Pants;
    USHORT          Dresses;
    WORD            Ties;

    cout << "Enter number of shirts: ";
    cin >> Shirts;
    cout << "Enter number of pants: ";
    cin >> Pants;
    cout << "Enter number of dresses: ";
    cin >> Dresses;
    cout << "Enter number of ties: ";
    cin >> Ties;
    cout << "\nCurrent Order"
         << "\n\tShirts: " << Shirts
         << "\n\tPants: " << Pants
         << "\n\tDresses: " << Dresses
         << "\n\tTies: " << Ties;

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

```

Here is an example of what it would produce:

```

The following pieces of information are required for each student
Day of Birth: 12
Month of Birth: 06
Year of Birth: 1988

Student's Date of Birth: 12/6/1988
Press any key to continue...

```

unsigned int, Cardinal, and UINT

The unsigned int, its Borland redefined Cardinal, and the Win32 UINT are used to identify a 32-bit positive integer variable whose value would range from 0 to 2147484647.

C++ Builder provides an effective checking process when you use the Cardinal or the USHORT, which improves the code. For example, if the user (or the program) is supposed to provide a positive number, such as the count of students in a classroom, but provides anything else than a positive integer, the compiler would reset the value of the variable to 1; consequently, no error (called exception) would



be raised. Therefore, when writing a program, it is safer to declare a positive integer as Cardinal or USHORT.

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    unsigned int Books;
    Cardinal NbrOfToes;
    USHORT NbrOfPeople;

    cout << "Estimate the number of books you have: ";
    cin >> Books;
    cout << "How many toes does a porc have? ";
    cin >> NbrOfToes;
    cout << "Enter the number of people who live with you: ";
    cin >> NbrOfPeople;
    cout << "\nYou have a collection of " << Books << " books";
    cout << "\nFrom the information you supplied, a porc has "
        << NbrOfToes << " toes.";
    cout << "\nYou live in a house of " << NbrOfPeople << " people.";

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

unsigned long, DWord, DWORD, and ULONG

When appended with a sign qualifier, the long, redefined as DWord, designates a positive 32-bit integer that ranges from 0 to 4294967295. When a variable requires a positive integer, you should use a DWord, a DWORD, or a ULONG identifiers because Borland C++ Builder checks that their values are positive, which avoids raising exceptions.

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    unsigned long USArea;
    DWord USPopulation;
    DWORD CdaArea;
    ULONG CdaPop;

    cout << "What is the area of the US? ";
    cin >> USArea;
    cout << "What is the population of the US? ";
    cin >> USPopulation;
    cout << "What is the area of Canada? ";
    cin >> CdaArea;
    cout << "What the population of Canada? ";
    cin >> CdaPop;

    cout << "\nCharacteristics of Canada";
    cout << "\n\tArea = " << CdaArea
        << "\n\tPopulation = " << CdaPop;
}
//-----
```

```
cout << "\nCharacteristics of the US";
cout << "\n\tArea = " << USArea
    << "\n\tPopulation = " << USPopulation;

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

## Using Integers

1. To use an example of an integer, change the content of the file as follows:

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

//-----

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

    Temperature = -8;
    cout << "The current temperature is " << Temperature << endl;

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

2. To test the program, press F9.
3. After testing the program, return to Bcb.
4. To use another technique of initializing a variable, make the following changes:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Temperature;
    Temperature = -8;
    int Width(24);

    cout << "The current temperature is " << Temperature << endl;
    cout << "The value of the width is " << Width << endl;

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

5. Test the program and return to Bcb.
6. To change the initialization of the Temperature variable, apply this change:

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

```
//-----
#pragma argsused
int main(int argc, char* argv[])
{
    int Temperature = -8;
    int Width(24);

    cout << "The current temperature is " << Temperature << endl;
    cout << "The value of the width is " << Width << endl;

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

7. Test the program.

8. To use a variant of an integer, change the file's content as follows:

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

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    unsigned int NumberOfStudents;

    cout << "How many students in this classroom? ";
    cin >> NumberOfStudents;
    cout << "\nThere are " << NumberOfStudents << " students.\n";

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

9. To test the program, press F9.

10. When asked to type a value, type -32 (which is negative) and press Enter.

11. Notice that the result is garbage.

12. Run the program again.

13. When asked to type a value, type 32 and press Enter.

14. After testing the program, return to Bcb.

15. To perform a basic sum in a program, make the following change:

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

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    unsigned int Grade1Students, Grade2Students;

    cout << "How many students in the classrooms?\n";
    cout << "Grade 1: ";
    cin >> Grade1Students;
    cout << "Grade 2: ";
```

```
cin >> Grade2Students;
cout << "\nNumber of students";
cout << "\nGrade 1: " << Grade1Students
    << "\nGrade 2: " << Grade2Students;
cout << "\nTotal: " << Grade1Students + Grade2Students << endl;

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

16. Test the program. Make sure you supply positive values.

17. To use a mixing of various integer types, make the following changes:

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

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    unsigned int Grade1Students, Grade2Students;
    short Total, Average;

    cout << "How many students in the classrooms?\n";
    cout << "Grade 1: ";
    cin >> Grade1Students;
    cout << "Grade 2: ";
    cin >> Grade2Students;

    Total = Grade1Students + Grade2Students;
    Average = Total / 2;

    cout << "\nNumber of students";
    cout << "\nGrade 1: " << Grade1Students
        << "\nGrade 2: " << Grade2Students;
    cout << "\nTotal: " << Total;
    cout << "\nAverage: " << Average << endl;

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

18. Test the program.

## Characters

A character is an individual symbol that displays on your screen. It could be a letter such as a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, and z; from A to Z, a digit such as 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9; or any other special characters such as ` ~ \$ ! @ % ^ & \* ( { [ ] } | \ : ; " ' + - < \_ ? > , / =. A character is really an 8-bit of integer whose value can range from -128 to 127.

To declare a character variable, you can use:

```
char AlphaLetter;
```

Since a char variable represents one symbol, to initialize it, enclose the initial value in single quotes. Here are examples:

```
char Answer = 'y';
char Pick('r');
```

char, signed char, AnsiChar, and CHAR

To declare a character variable, use either the char, the signed char, or the AnsiChar keywords. This type of variable would be an 8-bit integer whose value can range from -128 to +127.

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    char Satisfaction;
    signed char AgeCategory;
    AnsiChar Answer;
    CHAR Size;

    cout << "From A to Z, enter a character as your level of satisfaction: ";
    cin >> Satisfaction;
    cout << "Age category (t=teen/a=Adult/s=Senior): ";
    cin >> AgeCategory;
    cout << "Are you drunk (y=Yes/n=No)? ";
    cin >> Answer;
    cout << "Enter your size (s=Small/m=Medium/l=Large): ";
    cin >> Size;

    cout << "\nSatisfaction: " << Satisfaction;
    cout << "\nAge category: " << AgeCategory;
    cout << "\nYour answer: " << Answer;
    cout << "\nYour size: " << Size;

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

```
From A to Z, enter a character as your level of satisfaction: t
Age category (t=teen/a=Adult/s=Senior): a
Are you drunk (y=Yes/n=No)? n
Enter your size (s=Small/m=Medium/l=Large): l
```

```
Satisfaction: t
Age category: a
Your answer: n
Your size: l
```

Press any key to continue...

unsigned char, Char, Byte, ByteBool, BYTE, and UCHAR

To keep the integer value of a char positive, you can declare it as an unsigned char, a Char, a Byte, a ByteBool, a BYTE, or a UCHAR. Its value would then range from 0 to 255.

### ◆ Using Character Variables

1. To declare and initialize a constant character, change the listing of the file as follows:

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

```
#pragma hdrstop

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    char Pick = 'r';

    cout << "Pick " << Pick;

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

2. Test the program and return to Bcb.

3. To use other techniques of declaring and initializing character variables, change the content of the file as follows (the NewLine variable will insert a new line in the program; the Lotus variable will produce a sound (beep) when the program runs):

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    char Pick = 'r';
    char NewLine = '\n';
    unsigned char Lotus('\x007');

    cout << "Pick " << Pick;
    cout << NewLine;
    cout << "Lotus " << Lotus;

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

4. Test the program and return to Bcb.

## Arrays of Characters

A group of characters is called an array. This is a study we will cover when learning about arrays. For now, if you want to use a group of characters of any kind, declare a variable starting with a char data type, followed by a valid name for the variable, followed by an opening square bracket "[", followed by a number of characters, followed by a closing square bracket "]", and ended with a semi-colon.

The syntax to declare a group of characters is:

```
char VariableName[NumberOfCharacters];
```

A character data type is required to let the compiler know that you want to create a variable of type char. The created variable should have a valid name. The number of characters, called the dimension of the array, should be an estimate; for example, if you want to request employees' first names, type a number that you think would represent the largest name possible. The maximum number should be 80, but the average and a good regular number should be 12 or 20. Examples of declaring

arrays of characters are:

```
char FirstName[12];
char LastName[12];
```

To initialize an array of characters, do not specify the dimension and leave the square brackets empty. You can use the assignment operator and include the initialized variable in double-quotes. Another technique, is to include the value between parentheses. Here is a program with two techniques of initializing arrays of characters:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    char University[] = "University of the District of Columbia";
    char Faculty[]("Computer sciences");

    cout << "Welcome to the Student Orientation Program.\n";
    cout << "\nFor your studies, we have selected,";
    cout << "\nInstitution: " << University;
    cout << "\nFaculty:      " << Faculty;

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

You can use an array variable to request a group of words from the user. To request a group of words using an array, use one of the following functions:

```
cin.getline( VariableName, Dimension);
cin.getline(VariableName, Dimension, Delimeter);
```

## Strings

A string is a character, a group of characters, or an empty space that you want the compiler to treat "as is". A string is created using the string keyword. The biggest difference between a char and a string identifiers is that the char identifies a variable made of only one character while the string identifier almost has no limit.

To use a string in your program, declare a variable starting with the word string followed by a valid name for the variable. Here are examples:

```
string Continent;
string Company;
```

When requesting its value from the user, by default, the string identifier is used to get only a one-word variable. Here is an example program that requests a first and last names:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
```

```
    string FirstName, LastName;

    cout << "Enter first name: ";
    cin >> FirstName;
    cout << "Enter last name: ";
    cin >> LastName;
    cout << "\n\nFull Name: " << FirstName << " " << LastName;

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

You can initialize a string variable of any length. One technique is to use the assignment operator and include the string in double-quotes. Here is an example:

```
string UN = "United Nations";
```

```
cout << "The " << UN << " is an organization headed by a Secretary General";
```

Another technique involves using parentheses following the name of the string variable, and including the string in double-quotes. Here is an example:

```
string BookTitle("Drugs, Sociology, and Human Behavior.");
```

```
cout << "For our class next week, please read \"" << BookTitle; cout << "\"";
```

If you want to request the value of the variable from the user, you should use the getline function. To use the getline function, follow this formula:

```
getline(cin, StringName);
```

Inside of the parentheses, the word cin informs the compiler that the request will come from an external source, mainly the user typing from the keyboard. The StringName is the name you declared the variable with. The getline() function expects that the user will press Enter to end the sentence; the end line character is '\n'.

Here is an example program that requests strings of any length from the user:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    string MusicAlbum;
    string TrackTitle;

    cout << "Welcome to Radio Request where the "
         << "listeners select their songs:\n";
    cout << "Type the album name: ";
    getline(cin, MusicAlbum);
    cout << "Type the song title: ";
    getline(cin, TrackTitle);

    cout << "\nNow for your pleasure, we will play: "
         << TrackTitle << "\nfrom the "
         << MusicAlbum << " wonderful album.";

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

```

    return 0;
}
//-----

```

If you want the user to end the sentence with another character such as \* or !, use the following function

```
getline(cin, StringName, Delimiter);
```

The following example uses the = symbol as the end character of the sentence:

```
string Address;cout << "Enter your address. To end, type =\n";
getline(cin, Address, '=');cout << "\nSo, you live at: " << Address;
```

### ◆ Using Strings

1. Create a new project based on the Console Wizard
2. To apply the basic concepts we have learned about the string identifier, change the content of the file as follows:

```

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

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    string FirstName, LastName;

    cout << "Welcome to College Park Auto-Parts\n";
    cout << "Enter the following information about the customer's.\n";
    cout << "First Name: ";
    cin >> FirstName;
    cout << "Last Name: ";
    cin >> LastName;

    cout << "\n\nCPAP Invoice # 1202";
    cout << "\nCustomer Name: " << FirstName << " " << LastName;

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

```

3. To test the program, press F9.

## Floating-Point Variables

The integers we have used so far have the main limitation of not allowing decimal values. C++ provides floating identifier values that would solve this problem.

To declare a variable that involves a decimal value, use a floating-point identifier.

float, FLOAT, and Single

The most fundamental floating variable is created with the float keyword. The float is a 4 Byte real number that ranges from  $3.4 \times 10^{-38}$  to  $3.4 \times 10^{38}$ . To declare a floating variable, use the float, the (Win32) FLOAT, or the (Borland C++ Builder) Single keywords followed by the name of the variable. Examples are:

```
float Side;
FLOAT Length;
```

Single Width;

To initialize a floating variable, after declaring it, assign it a value using the Assignment operator, like this:

```
float PriceSoda;
PriceSoda = 0.85;
```

You can also initialize the variable using the parentheses, like this:

```
float PriceOrangeJuice(2.25);
```

To request a floating variable from the user, use the cin operator, like this

```
cin >> Variable;
```

Like an initialized variable, you can perform any desired operation on such a variable. Here is an example program that requests the side of a square then calculates the perimeter and the area:

```

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

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    float Side, Perimeter, Area;

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

    Perimeter = Side * 4;
    Area = Side * Side;

    cout << "\nCharacteristics of the square:";
    cout << "\nSide:      " << Side;
    cout << "\nPerimeter: " << Perimeter;
    cout << "\nArea:       " << Area;

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

```

### ◆ Using Floating-Point Variables

1. Create a new C++ application based on a Console Wizard.
2. Change the content of the file as follows:

```

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

//-----
#pragma argsused
int main(int argc, char* argv[])
{
    float PriceSoda = 0.85;
    float PriceOJ(2.25);
    float PriceSubs, PriceSandwich;

```

```

float Sodas, OJ, Subs, Sands, Total;

PriceSubs = 3.25;
PriceSandwich = 3.65;

int NbrOfSoda, NbrOfOJ, NbrOfSubs, NbrOfSands;

cout << "Process customer's order:\n";
cout << "Cans of Sodas: ";
cin >> NbrOfSoda;
cout << "Bottles of Orange Juice: ";
cin >> NbrOfOJ;
cout << "Number of Subs: ";
cin >> NbrOfSubs;
cout << "Number of Sandwiches: ";
cin >> NbrOfSands;
Sodas = NbrOfSoda * PriceSoda;

OJ = NbrOfOJ * PriceOJ;
Subs = NbrOfSubs * PriceSubs;
Sands = NbrOfSands * PriceSandwich;
Total = Sodas + OJ + Subs + Sands;

cout << "\nJanice's Corner - Customer Receipt -";
cout << "\nQty\tItem\tPrice\tTotal";
cout << "\n-----";
cout << "\n " << NbrOfSoda << "\tSoda\t" << PriceSoda << "\t" << Sodas;
cout << "\n " << NbrOfOJ << "\tOrange Jce\t" << PriceOJ << "\t" << OJ;
cout << "\n " << NbrOfSubs << "\tSubs\t" << PriceSubs << "\t" << Subs;
cout << "\n " << NbrOfSands << "\tSandwich\t"
    << PriceSandwich << "\t" << Sands;
cout << "\n-----";
cout << "\n\tTotal Order = $" << Total;
cout << "\n=====";

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

```

3. To test the application, press F9.

### double and Double

When a variable is larger than the float or the Single identifiers can handle and requires more precision, you should use the double or the Double identifiers. The double-precision identifier is an 8 Byte decimal or fractional number ranging from  $1.7 \times 10^{-308}$  to  $1.7 \times 10^{308}$ . It is used for numbers that and/or are very large.

```

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    Double ItemPrice, TaxCollected;
    double TaxRate(0.05);

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

    TaxCollected = ItemPrice * TaxRate;

    cout << "Tax amount collected for this item: $" << TaxCollected;
}

```

```

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

```

### long double and Extended

For an even larger variable, use the 10 Byte real data type identified as a long double, or an Extended variable that ranges from  $3.4 \times 10^{-4932}$  to  $1.1 \times 10^{4932}$ .

Here is an example that uses the Extended data type:

```

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    Extended radius = 15.625,
              Radius = 18.125;
    Extended area, Area, TotalArea;

    Area = Radius * Radius * 3.14159;
    area = radius * radius * 3.14159;
    TotalArea = Area - area;

    cout << "Properties of the plate";
    cout << "\nExternal Radius: " << Radius;
    cout << "\nInternal Radius: " << radius;
    cout << "\nTotal Area:      " << TotalArea;

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

```

## References

A reference is a variable name that is a duplicate of an existing variable. It provides a techniques of creating more than one name to designate the same variable. The syntax of creating or declaring a reference is:

```
DataType &ReferenceName = VariableName;
```

Therefore, to declare a reference, type the variable's name preceded by the same type as the variable it is referring to. Between the data type and the reference name, type the ampersand operator "&". To specify what variable the reference is addressed to, use the assignment operator "=" followed by the name of the variable. The referred to variable must exist already. You cannot declare a reference as:

```
int &Mine;
```

The compiler wants to know what variable you are referring to. Here is an example:

```

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

```

```
//-----
#pragma argsused
int main(int argc, char* argv[])
{
    int Number = 228;
    int &Nbr = Number;

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

The ampersand operator between the data type and the reference can assume one of three positions as followed:

```
int& Nbr;
int & Nbr;
int &Nbr;
```

As long as the & symbol is between a valid data type and a variable name, the compiler knows that the variable name (in this case Nbr) is a reference.

Once a reference has been initialized, it holds the same value as the variable it is pointing to. You can then display the value of the variable using either of both:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Number = 228;
    int & Nbr = Number;

    cout << "Number = " << Number << "\n";
    cout << "Its reference = " << Nbr;

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

If you change the value of the variable, the compiler updates the value of the reference so that both variable would hold the same value. In the same way, you can modify the value of the reference, which would update the value of the referred to variable. To access the reference, do not use the ampersand operator; just the name of the reference is sufficient to the compiler. This is illustrated in the following:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Number = 228;    // Regular variable
```

```
int& Nbr = Number; // Reference

cout << "Number = " << Number << "\n";
cout << "Its reference = " << Nbr << "\n";

// Changing the value of the original variable
Number = 4250;
cout << "\nNumber = " << Number;
cout << "\nIts reference = " << Nbr << "\n";

// Modifying the value of the reference
Nbr = 38570;
cout << "\nNumber = " << Number;
cout << "\nIts reference = " << Nbr;

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

In the way, you can use either a reference or the variable it is referring to, to request the variable's value from the user. Here is an example:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    double Price;
    double& RefPrice = Price;

    cout << "What's the price? $";
    cin >> Price;
    cout << "Price = $" << Price << "\n";
    cout << "Same as $" << RefPrice << "\n\n";

    cout << "What's the price? $";
    cin >> RefPrice;
    cout << "Price = $" << Price << "\n";
    cout << "Same as $" << RefPrice << "\n";

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

## The typedef Type Definition

Some of the identifiers we have used so far and some others we will learn further with arrays, pointers, and objects can be simplified and reduced to a new one-word identifier. C++ allows you to define a new variable using typedef. Indeed, typedef is not a new or other identifier. It provides a technique of redefining any of the known identifiers or by naming those that are combined.

The formula of using the typedef is:

```
typedef Identifier NewName;
```

The typedef word is required to let the compiler know that you are creating a new identifier. The Identifier is any of those we have learned so far. It could be an int,

an unsigned int, a char, a double, etc. An example of declaring a typedef is:

```
typedef int NumberOfStudents;
```

In this case, NumberOfStudents is just a new name for an int. It can be used as a new identifier exactly as if you were using an int. Here is an example that redefines an int identifier:

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

//-----

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

    NumberOfStudents Grade1, Grade2;
    cout << "Enter the number of students.\n";
    cout << "Grade 1: ";
    cin >> Grade1;
    cout << "Grade 2: ";
    cin >> Grade2;

    cout << "\nNumber of students:";
    cout << "\n1st Grade: " << Grade1;
    cout << "\n2nd Grade: " << Grade2;

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

Here are examples of creating new identifiers using typedef:

```
typedef unsigned int uInt;
typedef unsigned char uChar;
typedef unsigned long uLong;
typedef long double lDouble;
```

The compiler you are using shipped with a lot of data types, some of which are redefined versions of existing C++ data types. Therefore, the identifiers you will be using in C++ Builder come in three sets. The C++ language has its own data types that are integers, floating values, etc. Borland C++ Builder has a list of redefined types in the sysmac.h library. The Microsoft Windows operating systems ships with the Win32 library that has a set of redefined data types. To see its list, in the MSDN library, do a search on Data Types [Win32].

In this book, we will interchangeably use any data type we see fit for the task at hand, without identifying whether it is part of C++, Borland C++ Builder, or Win32. For that reason, from now on, many of our programs will include the VCL library.

### ◆ Using the typedef Keyword

1. To open WordPad, from the Taskbar, click Start -> Programs -> Accessories -> WordPad.
2. Locate the C:\Program Files\Borland\CBuilder5\Include\Vcl folder.
3. Double-click sysmac to open that header file.
4. Scroll down to locate the section with the type definitions.

5. Close WordPad. If you are asked to save anything, click No.

## Incrementing Values

We are used to counting numbers such as 1, 2, 3, 4, etc. In reality, when counting such numbers, we are only adding 1 to a number in order to get the next number in the range. C++ provides a technique of transparently counting such numbers.

### Incrementing a number

The simplest technique of incrementing a value consists of adding 1 to a variable. After adding 1, the value or the variable is (permanently) modify and the variable would hold the new value. This is illustrated in the following example:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Value = 12;

    cout << "Value = " << Value << endl;
    Value = Value + 1;
    cout << "Value = " << Value << endl;

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

C++ provides a special operator that takes care of this operation. The operator is called the increment operator and is ++. Instead of writing Value = Value + 1, you can write Value++ and you would get the same result. The above program can be re-written as follows:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Value = 12;

    cout << "Value = " << Value << endl;
    Value++;
    cout << "Value = " << Value << endl;

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

The increment++ is called a unary operator because it operates on only one variable. It is used to modify the value of the variable by adding 1 to it.



Every time the Value++ is executed, the compiler takes the previous value of the variable and adds 1 to it and the variable holds the incremented value:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Value = 12;

    cout << "Value = " << Value << endl;
    Value++;
    cout << "Value = " << Value << endl;
    Value++;
    cout << "Value = " << Value << endl;
    Value++;
    cout << "Value = " << Value << endl;

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

## Pre and Post-Increment

When using the ++ operator, the position of the operator with regard to the variable it is modifying can be significant. To increment the value of the variable before re-using it, you should position the operator on the left of the variable:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Value = 12;

    cout << "Value = " << Value << endl;
    cout << "Value = " << ++Value << endl;
    cout << "Value = " << Value << endl;

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

When writing ++Value, the value of the variable is incremented before being called. On the other hand, if you want to first use a variable, then increment it, in other words, if you want to increment the variable after calling it, position the ++ operator on the right side of the variable:

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

```
//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Value = 12;

    cout << "Value = " << Value << endl;
    cout << "Value = " << Value++ << endl;
    cout << "Value = " << Value << endl;

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

## Decrementing – Pre and Post-Decrementing

When counting numbers backward, such 8, 7, 6, 5, etc, we are in fact subtracting 1 from a value in order to get the lesser value. This operation is referred to as decrementing a variable. This operation works as if a variable called Value has its value diminished by 1, as in

Value = Value – 1;

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Value = 8;

    cout << "Value = " << Value << endl;
    Value = Value - 1;
    cout << "Value = " << Value << endl;

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

As done with the increment, C++ provides a quicker way of subtracting 1 from a variable. This is done using the decrement operation, that is --. To use the decrement operator, type – on the left or the right side of the variable when this operation is desired. Using the decrement operator, the above program could be written:

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Value = 8;

    cout << "Value = " << Value << endl;
```

```

Value--;
cout << "Value = " << Value << endl;

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

```

Once again, the position of the operator is important. If you want to decrement the variable before calling it, position the operator on the left side of the operator. This is illustrated in the following program:

```

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Value = 8;

    cout << "Value = " << Value << endl;
    cout << "Value = " << --Value << endl;
    cout << "Value = " << Value << endl;

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

```

If you plan to decrement a variable only after it has been accessed, position the operator on the right side of the variable. Here is an example:

```

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    int Value = 8;

    cout << "Value = " << Value << endl;
    cout << "Value = " << Value-- << endl;
    cout << "Value = " << Value << endl;

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

```

## Miscellaneous Increment and Decrement

It is not unusual to add or subtract a constant value to or from a variable. All you have to do is to declare another variable that would hold the new value. Here is an example:

```

//-----
#include <vcl.h>

```

```

#include <iostream.h>
#pragma hdrstop

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    double Value = 12.75;
    double NewValue;

    cout << "Value = " << Value << endl;
    NewValue = Value + 2.42;
    cout << "Value = " << NewValue << endl;

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

```

The above technique requires that you use an extra variable in your application. The advantage is that each value can hold its own value although the value of the second value depends on whatever would happen to the original or source variable. Sometimes in your program you will not need to keep the value of the source variable. You might want to simply permanently modify the value that a variable is holding. In this case you can perform the addition operation directly on the variable by adding the desired value to the variable. This operation modifies whatever value a variable is holding and does not need an additional variable. To add a value to a variable and change the value that the variable is holding, use the assignment "=" and the addition "+" operators as follows:

```

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    double Value = 12.75;

    cout << "Value = " << Value << endl;
    Value = Value + 2.42;
    cout << "Value = " << Value << endl;

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

```

To diminish the value of a variable, instead of the addition operation, use the subtraction and apply the same technique. In the above program, the variable can have its value decremented by applying the assignment and the subtraction operations on the variable as follows:

```

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

//-----

#pragma argsused
int main(int argc, char* argv[])

```

```

{
    double Value = 12.75;

    cout << "Value = " << Value << endl;
    Value = Value - 5.36;
    cout << "Value = " << Value << endl;

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

```

## Constants

A constant is a value that does not change. There are various categories of constants you will be using in your programming life: Those that are mathematically defined (such as numbers); those that are defined by, and are part of, the C++ language; those defined by the Microsoft Windows operating system you are using, and those defined by the Borland C++ Builder compiler or libraries. To make their management easier, these constant values have been categorized and defined in particular libraries. Throughout this book, you will be using them.

## Constant Values

The algebraic numbers you have been using all the time are constants because they never change. Examples of constant numbers are 12, 0, 1505, or 88146. Therefore, any number you can think of is a constant.


Every letter of the alphabet is a constant and is always the same. Examples of constant letters are d, n, c.

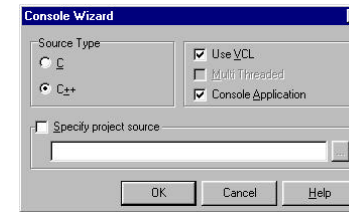
Some characters on your keyboard represent symbols that are neither letters nor digits. These are constants too. Examples are #, &, |, !.

Some values would be constant by default, but their constancy sometimes depends on the programmer. For example, one programmer can define a const PI as 3.14; another programmer can decide that the constant PI would be 3.14159; yet another programmer would use a more precise value. Therefore, you will be defining your own constant values as you see fit for the goal you are trying to achieve.

There are two main techniques you use to display a constant value in C++. To simply display it using the cout operator, you can use its value on the right side of the << symbols. You can also define it first using an appropriate name, and then using that name to display the constant.

### ◆ Using Constant Values

1. On the Standard toolbar, click the New button .
2. On the New Items dialog, click Console Wizard and click OK
3. On the Console Wizard dialog, make sure that the C++ radio button, the VCL, and the Console Application check boxes are selected:



4. Click OK
5. Change the content of the file as follows:

```

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

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    cout << 28;
    cout << "\nStudent Age: " << 14;

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

```

6. To test the program, press F9.

The safest technique of using a constant is to give a name. This allows you to manage it from one standpoint. For example, if you plan to use a number such as 3.14 that represents PI, you can simply use the constant 3.14. Imagine you want to use 3.14 in various sections of the program such as in different functions. If you decide to change the number from 3.14 to 3.14159 or another value, you would have to find every mention of 3.14; this is cumbersome and considered bad programming. The alternative is to create a variable and assign it the desired value. That new and defined value is called a constant.

There are two main techniques used to create constant values. The old way, which was used with the C language and widely used in documentation and help file consists of using the define keywords. The formula of creating a constant using the define word is:

```
#define ConstantName ConstantValue
```

The # symbol and the define word are required; they inform the compiler to consider that the following name represents a constant. The ConstantName represents a valid name for the desired constant; the name follows the same rules we learned for defining names in C++. To distinguish the constant from other names of variables, it is a good to write it in uppercase, although it is perfectly allowed in lowercase or any case combination you desire. The ConstantValue is a character, integer, floating-point value, or expression. If the constant value is an integer or a floating-point value, you can type. If the value is a character, include it between single-quotes. The definition of the constant does not end with a semi-colon. Examples of declaring constants are:

```
#define AGE 12 // AGE represents the constant integer 12
#define ANSWER 'y'
```

```
#define MAXSTUDENTS 35
#define PI 3.14159 // PI represents 3.14159
```

Another technique of creating a constant is by using the const keyword. The formula of using the const keyword is:

```
const ConstantName = ConstantValue;
Or
const Identifier ConstantName = Value;
```

The const keyword is required to inform the compiler that you are defining a constant. The ConstantName specifies the name of the constant value; it follows the rules we have applied for the names of variables. The Identifier, although optional, is used to let the compiler know what kind of integer it is. If you don't specify an identifier, the compiler will assume that the constant is an integer. Therefore, make it a habit to always specify the identifier when creating a constant using the const keyword. The identifier is any of the identifiers we have learned. Use the assignment operator to assign the desired constant to the name.

Examples of creating constants with the const keyword are:

```
const float PI = 3.14159;
const unsigned int MaxStudents = 42;
const string Country = "New Zealand";
const double Distance = 1678212;
```

## Operating System and Compiler Constants

Another category of constants are those that are part of the C++ compiler or of the Microsoft Windows operating system. Those constants have names they can be recognized with. For example, the name minimum value of the short integer is SHRT\_MIN; in the same way, the maximum short integer is identified by SHRT\_MAX. You can use either of these constants as follows:

```
//-----
#include <vc1.h>
#include <iostream.h>
#pragma hdrstop

//-----

#pragma argsused
int main(int argc, char* argv[])
{
    cout << "The minimum short integer is: " << SHRT_MIN << "\n";
    cout << "The maximum short integer is: " << SHRT_MAX << "\n";

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

The integral constants are defined in the \_lim.h library.

### ◆ Using Compiler Constants

1. To view the list of constant integers, open Windows Explorer.
2. Display the content of the C:\Program Files\Borland\Cbuilder\Include folder
3. Double-click the \_lim.h file to open it.  
Since you get the last way, I will not provide

---

[Previous](#)
[Copyright © 2002 FunctionX](#)
[Next](#)


---