Chapter II

Structure Data Types

In this chapter we will cover:
1. Type Definition
2. Enumerated Data Type
3. Structures

2.1 Type Definition in C++

The syntax for type definition is as follows:

typedef *known type* *new type***;**

Example:

```
typedef    unsigned char    BYTE;
```

For arrays the syntax is:

typedef *base type* *ArrayName***[***Array Size***];**

Example:

```
typedef   double   Matrix[10][10];

void main( )
  {
Matrix A, B;            //A and B are matrices of sizes 10×10
      .
      .
      .
  }
```

The following program illustrates the use of **typedef**

```
#include <iostream.h>
typedef double REAL8;
typedef double VECTOR[5];
typedef double MATRIX[2][2];

//------------------------------------------------------------------------
int main()
{
     REAL8 sum;
```

```
    VECTOR X={1, 2.0, 3.0, 4.5, 55.6};
    MATRIX A;
    int i,j;

    //Sum the values in X
    sum=0;
    for(i=0; i<5; i++)
      sum+=X[i];

    cout << sum << endl;

    //Enter a matrix
    cout << endl << "Enter a 2x2 matrix: " << endl;

    //Print-out matrix
    for(i=0; i < 2; i++)
     for(j=0; j < 2; j++)
       cin >> A[i][j];

    for(i=0; i < 2; i++)
     {
     for(j=0; j < 2; j++)
        cout << A[i][j] << " ";

     cout << endl;
     }
    return 0;
}
```

2.2 Visibility of variables

Variables declared before main() are called global variables and are visible throughout the main function and other proceeding functions. Variable declared within main() or any other function are called local variables and are visible within main or the function in which they are declared and only forward from the place in which they have been declared. Variables declared within a block are visible only within that block. The following code segments illustrates visibility of variables.

```
#include <iostream.h>

double X;              // global variable

void main()
 {
```

```
int j;                  //local variable visible within main
  .
  .
  .
if(...)
  {
   int k;       // visible only within the block defined within the two braces { }
     .
     .
     .
   }
k=3;        // Error - k not visible
  .
  .
  .
int i;   // local variable visible visible from that statement onwards
  .
  .
  .
}
```

2.3 Enumerated Data Types

An enumerated data type is used to provide mnemonic identifiers for a set of integer values. For example, the following declaration:

```
enum WeekDays { sun, mon, tues, wed, thur, fri, sat };
```

establishes a unique integral type, WeekDays.

To use WeekDay you will have to declare a variable of type WeekDays:

```
WeekDays    Days;
```

Now Days can be assigned the value: sun or mon or tue , etc.

The enumerator constants inside the braces are assigned fixed integral values. (sun) is assigned zero, mon=1, etc. Each succeeding enumerator constant is set one more than its predecessor. The pre-assigned values can be overridden by the user as follows:

```
enum WeekDays { sun=1, mon, tues, wed, thur, fri, sat };
```

Now sun=1, mon=2, tues=3, etc.

Other examples of enum declarations:

```
enum colors { black=1, red=2, blue=3, green=5, yellow=7, white=11 };

enum BOOL { False,  True };


/* Initializer expression can include previously declared enumerators */
enum coins { penny=1, nickel =  penny +4, dime = 10, quarter = nickel*nickel,
             two_bits = quarter };
```

the declaration of the enumerator type may include the declaration of that type:

```
enum WeekDays { sun=1, mon, tues, wed, thur, fri, sat } PayDay, MovieDay;
```

In C++, a variable of an enumerated type can be assigned only one of its enumerators. That is,

```
PayDay = fri;      //OK
MovieDay = 7;   // illegal, even though sat = = 7;
```

The following example demonstrates usage of enumerator types:

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

//---------------------------------------------------------------------------
enum WeekDays {Sunday=1, Monday, Tuesday, Wednesday, Thursday, Friday,
     Saturday };
//enum  BOOL {false,true}; // not required in C++Builder

int main()
{
  WeekDays day;
  BOOL more;
  char akey;
  unsigned j;

  do {
    do {
    cout << "Enter a day number (Sun=1, Mon=2, etc.):" ;
    cin >> j;
      } while (j<1 || j>7);
    day=WeekDays(j);
```

```
    switch(day)
     {
     case Sunday:
     case Saturday:
        cout << "Weekend!";
        break;
     case Monday:
     case Tuesday:
     case Wednesday:
     case Thursday:
        cout << "Work, work, work!";
        break;
     case Friday:
        cout << "T.G.I.F. !!";

     }
     cout << "\n more? (y/n) ";
     cin >> akey;
     if(akey == 'Y'  || akey == 'y')
        more = true;
     else
        more = false;
     cout << endl << endl;
      } while (more == true
      );
   getch(); // wait for a key to be pressed before
            // terminating the program.
   return 0;
}
```

The above program should be relatively easy to follow. The user is asked to enter a specific day in numbers, where a 1 entered means Sunday, and so on. The program uses the do ... while and the switch structures discussed in Chapter I.

Note that the statements:

```
     if(akey == 'Y'  || akey == 'y')
        more = true;
     else
        more = false;
```

could have been written as:

```
     more = (akey == 'Y'  || akey == 'y') ? true : false;
```

In general:

```
variable = (condition) ? value1 : value2;
```

Is the same as:

```
if(condition = = true)
        variable = value1;
else
        variable = value2;
```

2.4 Character constants

Character constants are specified in single quotes; for example 'a', 'f', 'w'. The backslash character (\ ) is used to introduce an *escape sequence*, which allows the visual representation of certain non-graphic characters. For example, the constant, '\n' is used to specify a new line (line feed) and '\t' is used to represent a horizontal tab.

Characters are represented internally by 8 bit numerical codes. One of these codes that is used most often in C++ is the ASCII (American Standard Code for Information Interchange). For example, the ASCII representation of 'a' is 97 (in decimal representation). You can obtain the ASCII decimal or hexadecimal code using a simple program as the one shown next.

```
#include <iostream.h>

int main()
{
char ch;
  while(1)
    {
  cout << "Enter a character --> ";
  cin >> ch;
  cout << dec << int(ch) << ' ';
  cout << hex << int(ch) << ' ';
  cout << ch << endl;
  cout << "Continue? (y/n) : ";
  cin >> ch;
  cout << endl;
  if(ch=='n' || ch =='N') break;
    }
  return 0;
}
```

Examples of escape sequences are listed in Table 2.1

Table 2.1 Escape sequences

Sequence	Value	Char	?
\a	0x07	BEL	Audible bel
\b	0x08	BS	Backspace
\f	ox0C	FF	Form feed
\n	0x0A	LF	New line
\r	0x0D	CR	Carriage return
\t	0x09	HT	Horizontal tab
\v	0x0B	VT	Vertical tab
\\	0x5c	\	Backslash
\’	0x27	‘	Single quote
\”	0x22	“	Double quote
\?	0x3F	?	Question mark
\O		any	O=a string of up to three octal digits.
\xH \XH		any	H=a string of hex digits.
\101	octal 101	‘A’	
\x041	hexadecimal 41	‘A’	
\0			null character

2.5 String constants

String constants are presented between double quotes. For example:

```
char str1[ ]= "John";
```

is stored as follows: 'J', 'o', ‘h’, ‘n’, ‘\0'. A string constant is always terminated with the null character \0. The size of str1 is 4+1, that is if we were to dimension str1 we would write the above statement as follows:

```
char str1[5 ]="John";
```

String constants or characters cannot be assigned in a program using a simple assignment

statement:

```
str1 = "John";    //Error
```

Assignment of string constants can be carried out during declaration using a simple assignment statement as shown above or the built library function **strcpy** which is prototyped in string.h:

```
strcpy(str1, "John");
```

2.6 Structures

A structure is a user defined type representing a collection of named members or components. Components can be of any built in or user defined types. The syntax for a structure is as follows:

```
struct   structTag {
                Type1 member1;
                Type2 member2;
                    .
                    .
                    .
              };
```

Example of structures:

```
struct complex {
        double real;
        double imag;
                };

struct personal {
        char name[30];
        char title[20];
        char address[31];
        char city[16];
        char province[4];
        char postalCode[8];
        unsigned age;
        char height[20];  // x feet y inches
        unsigned weight;   // in lbs
                };
```

C++ allows untagged structures. For example:

```
struct {
        double real;
```

```
        double imag;
        } c1, c2, c3;
```

C++ allows you to declare and initialize a structure variable:

```
complex c={5.0, -6.7};
personal Murphy = { "John Murphy","Mr.", "289 Sunset Ave.", "Windsor",
                    "N9B 3P4", 40, "5 feet 8 inches", 175};
```

To access members of a structure use the dot operator.

```
c1.real=3.0;
c1.imag=6.7;
Murphy.weight+=5; // gained 5 lbs.
```

The following program illustrates the use of structures:

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

#define sqr(x) ((x)*(x))   //Macro

struct coord {
        double x;
        double y;
        };

int main()
{
coord p1, p2;
double dx, dy, distance;

cout << "Enter x and y coordinates of two points --> ";
cin >> p1.x >> p1.y >> p2.x >> p2.y;
dx=p1.x-p2.x;
dy=p1.y-p2.y;
distance = sqrt(sqr(dx)+sqr(dy));
cout << "Distance = " << distance << endl;
getch();
      return 0;
}
```

Problems

1. Complete the following program:

```
#include <iostream.h>

typedef double REAL8;
typedef double MATRIX[3][3];

int main()
  {
REAL8 sum;
MATRIX A;

// Enter any 3x3 matrix
...

// Sum up all values in A
...
// Print matrix and sum
....
}
```

2. Given:

```
enum coins { penny=1, nickel =  penny +4, dime = 10, quarter = nickel*nickel,
                        two_bits = quarter };
```

Develop a program that breaks any dollar amount, e.g. $22.67 into whole dollars, quarters, dime, nickels and pennies.

3. Given the following structure:

```
struct employee {
        char name[30];
        char address[30];
        char city[30];
        char postalCode[10];
        float salary;
      } emps[5];
```

Develop a program that allows the user to enter information according to the above structure for 5 employees and then prints the entered information out.
(Note: You can enter a character string using the member function **cin.getline**(*String variable, size of string variable*);

e.g.
```
char str1[15];
cout << "Enter some input " << endl;
cin.getline(str1,15);
cout << str1 << endl;          )
```

6. Using a structure of the form:

```
Struct _Complex
{
double Real, Imag;
};
```

Develop a function that returns the absolute value of a complex number. You can declare the function as follows:

```
double cabs(_Complex A)
        {

        }
```

Note that: $|A| = \sqrt{x^2 + y^2}$; where $A = x + iy$