Chapter 5c

STRUCTURED TYPE

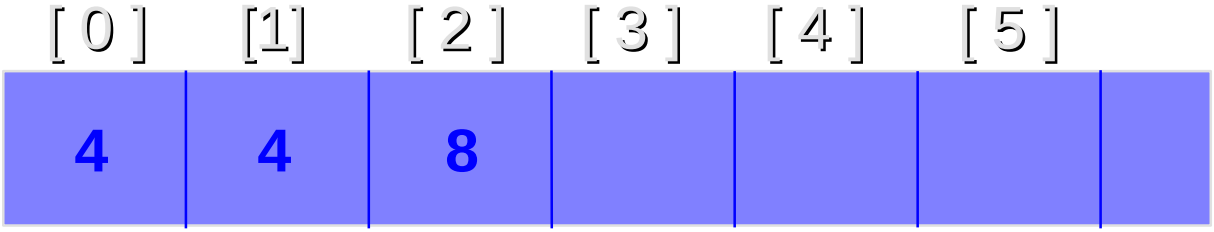
Chapter 5

- **Array type**
- **String type**
- **Structure type**

ARRAYS

- **An *array* is an advanced data type that contains a set of data represented by a single variable name.**
- **An element is an individual piece of data contained in an array.**
- **The following figure shows an integer array called *c*.**

c[0] = 4; c[1] = 4, c[2] = 8, etc.



Array Declaration

- **The syntax for declaring an array is**

  type name[elements];

- **Array names follow the same naming conventions as variable names and other identifiers.**
- **All elements of a C/C++ array must have the same type**

- **Example:**

```
int arMyArray[3];
char arStudentGrade[5];
```

- **The first declaration tells the compiler to reserve 3 elements for integer array *arMyArray*.**

Subscript

- **The numbering of elements within an array starts with an index number of 0.**

- **An *index number* is an element's numeric position within an array. It is also called *a subscript*.**

- **<u>Example</u>:**

***StudentGrade[0]* refers to the 1st element in the *StudentGrade* array.**
***StudentGrade[1]* refers to the 2nd element in the *StudentGrade* array.**
***StudentGrade[2]* refers to the 3rd element in the *StudentGrade* array.**
***StudentGrade[3]* refers to the 4th element in the *StudentGrade* array.**
***StudentGrade[4]* refers to the 5fth element in the *StudentGrade* array.**

A example of array

Example 5.8.1

```
#include <iostream.h>
int main(){
    char arStudentGrade[5]= {‘A’, ‘B’, ‘C’, ‘D’, ‘F’};
    for (int i = 0; i <5; i++)
       cout << arStudentGrade[i] << endl;
  return 0;
}
```

The output is:

```
A
B
C
D
F
```

Example 5.8.2

```
// Compute the sum of the elements of the array
#include <iostream>
int main()
{
   const int arraySize = 12;
   int a[ arraySize ] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
   int total = 0;
   for ( int i = 0; i < arraySize; i++ )
      total += a[ i ];
   cout << "Total of array element values is " << total << endl;
   return 0 ;
}
```

The output of the above program is as follows :

Total of array element values is 383

Multi-Dimensional Arrays

- **C++ allows arrays of any type, including arrays of arrays. With two bracket pairs we obtain a *two-dimensional* array.**
- **The idea can be iterated to obtain arrays of higher dimension. With each bracket pair we add another dimension.**

- **Some examples of array declarations**

```
int a[1000];          // a one-dimensional array
int b[3][5];          // a two-dimensional array
int c[7][9][2];       // a three-dimensional array
```

In these above examples, *b* has 3×5 elements, and *c* has $7 \times 9 \times 2$ elements.

A two-dimensional array

- **Starting at the base address of the array, all the array elements are stored contiguously in memory.**
- **For the array *b*, we can think of the array elements arranged as follows:**

	col 1	col2	col3	col4	col5
row 1	b[0][0]	b[0][1]	b[0][2]	b[0][3]	b[0][4]
row 2	b[1][0]	b[1][1]	b[1][2]	b[1][3]	b[1][4]
row 3	b[2][0]	b[2][1]	b[2][2]	b[2][3]	b[2][4]

Example 5.8.3 This program checks if a matrix is symmetric or not.

```
#include<iostream.h>
#include<iomanip.h>
const int N = 3;
void main( )
{
   int i, j;
   int a[N][N];
   bool symmetr = true;
   for ( i=0; i< N; ++i)
      for (j=0; j<N; ++j)
          cin >> a[i][j];
for(i= 0; i<N; i++){
          for (j = i+1; j < N; j++){
            if(a[i][j] != a[j][i]){
               symmetr = false;
               break;
            }
          }
          if(!symmetr)
             break;
    }
    if(symmetr)
      cout<<"\nThe matrix is symmetric“
           << endl;
    else
      cout<<"\nThe matrix is not symmetric“
            << endl;
   return 0;
}
```

Strings and String Built-in Functions

- **In C we often use character arrays to represent strings. A string is an array of characters ending in a null character (‘\0’).**
- **A string may be assigned in a declaration to a character array. The declaration**

```
char strg[] = “c”;
```

- **initializes a variable to the string “c”. The declaration creates a 2-element array *strg* containing the characters ‘c’ and *‘\0’*. The null character (\0) marks the end of the text string.**
- **The above declaration determines the size of the array automatically based on the number of initializers provided in the initializer list.**

- **In C, you must use a string built-in functions to manipulate *char* variables. Some commonly used string functions are listed in Table 5.1.**

Tabe 5.1 Common string functions

Function	Description
strcat(s1,s2)	Append one string to another
strchr(s1,a)	Find the first occurrence of a specified character in a string
strcmp(s1,s2)	Compare two strings
strcpy(s1,s2)	Replaces the contents of one string with the contents of another
strlen(s1)	Returns the length of a string

- **The *strcpy()* function copies a literal string or the contents of a *char* variable into another *char* variable using the syntax:**

  strcpy(destination, source);

  where *destination* represents the char variable to which you want to assign a new value to and the *source* variable represents a literal string or the char variable contains the string you want to assign to the destination.

- **The *strcat()* function combines two strings using the syntax:**

  strcat(destination, source);

  where *destination* represents the char variable whose string you want to combine with another string. When you execute *strcat()*, the string represented by the source argument is appended to the string contained in the destination variable.

Example:

```
char FirstName[25];
char LastName[25];
char FullName[50];
strcpy(FirstName, “Mike”);
strcpy(LastName, “Thomson”);
strcpy(FullName, FirstName);
strcat(FullName, “ “);
strcat(FullName, LastName);
```

- **Two strings may be compared for equality using the *strcmp()* function.** When two strings are compared, their individual characters are compared a pair at a time. If no differences are found, the strings are equal; if a difference is found, the string with the first lower character is considered the smaller string.

- **The function listed in Table 5.1 are contained in the *string.h* header file. To use the functions, you must add the statement *#include<string.h>* to your program.**

Example 5.8.4

```
#include<iostream.h>
#include<string.h>
int main()
{
    char FirstName[25];
    char LastName[25];
    char FullName[50];
    strcpy(FirstName, "Mike");
    strcpy(LastName, "Thomson");
    strcpy(FullName, FirstName);
    strcat(FullName, " ");
    strcat(FullName, LastName);
    cout << FullName << endl;
    int n;
    n = strcmp(FirstName, LastName);
    if(n<0)
        cout<< FirstName << " is less than "<< LastName<<endl;
```

```
else if(n ==0)
    cout<< FirstName << " is equal to “
          << LastName<<endl;
else
    cout<< FirstName << " is greater than “
        << LastName<<endl;
return 0;
}
```

The output of the program:

Mike Thomson

Mike is less than Thomson

STRUCTURES

- **A *structure*, or *struct*, is an advanced, user-defined data type that uses a single variable name to store multiple pieces of related information.**

- **The individual pieces of information stored in a structure are referred to as *elements, field*, or *members*.**

- **You define a structure using the syntax:**

```
struct struct_name{
data_type field_name;
data_type field_name;
……..
} variable_name;
```

To access a field inside a structure

- **Example:**

```
struct emloyee{
    char firstname[25];
    char lastname[25];
    long salary;
};
```

- **To access the field inside a structure variable, you append a period to the variable name, followed by the field name using the syntax:**

    variable.field;

- **When you use a period to access a structure fields, the period is referred to as the *member selection operator*.**

Example 5.9.1

```
#include <iostream.h>
struct Date    // this is a global declaration
{
  int month;
  int day;
  int year;
};
int main(){
  Date birth;   // birth is a variable belonging to Date type
  birth.month = 12;
  birth.day = 28;
  birth.year = 1982;
  cout << "\nMy birth date is "
       << birth.month << '/‘ << birth.day   << '/'
       << birth.year % 100 << endl;
  return 0;
}
```

Arrays of Structures

- **The real power of structures is realized when the same structure is used for lists of data.**

- **Declaring an array of structures is the same as declaring an array of any other variable type.**

- **Example 5.9.2:**

  The following program uses array of employee records. Each of employee record is a structure named *PayRecord*. The program displays the first five employee records.

```
#include <iostream.h>
#include <iomanip.h>
const int MAXNAME = 20;      // maximum characters in a name
```

```
struct PayRecord      // this is a global declaration
{
  long id;
  char name[MAXNAME];
  float rate;
};
int main()
{
  const int NUMRECS = 5; // maximum number of records
  int i;
  PayRecord employee[NUMRECS] = {
                        { 32479, "Abrams, B.", 6.72 },
                        { 33623, "Bohm, P.", 7.54},
                         { 34145, "Donaldson, S.",  5.56},
                         { 35987, "Ernst, T.", 5.43 },
                         { 36203, "Gwodz, K.", 8.72 } };
   cout << endl;   // start on a new line
```

```
cout << setiosflags(ios::left);
                        // left justify the output
 for ( i = 0; i < NUMRECS; i++)
   cout << setw(7)  << employee[i].id
     << setw(15) << employee[i].name
     << setw(6)  << employee[i].rate << endl;
return 0;
 }
```

The output of the program is:

32479	Abrams, B.	6.72
33623	Bohm, P.	7.54
34145	Donaldson, S.	5.56
35987	Ernst, T.	5.43
36203	Gwodz, K	8.72

Summary

- **Structured type contains many elements**
- **There are 3 structured types concerned in this lecture:**
  - Array type:
    - all elements have the same type
    - Each element can be accessed by index
  - String type:
    - Like array type but element type is char
    - Has extra (last) element that contains '\0'
  - Struct type:
    - Elements may be in different type
    - Each element can be accessed by name