To:
Tenouk

MODULE 11
THE C/C++ TYPE SPECIFIERS 1
struct, typedef, enum, union

My Training Period: xx hours

From this Module you can jump to the Object Oriented idea and C++ or proceed to extra C Modules or Microsoft C: implementation specific to experience how C is used in the real implementation. The struct lab worksheets are: C/C++ struct part 1 and C/C++ struct part 2. Also the combination of the struct, arrays, pointers and function C worksheet 1, C lab worksheet part 2 and C lab worksheet part 3.

C & C++ abilities that must be acquired:

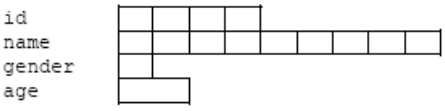
- Able to understand and use structure (struct).
- Able to relate structure, functions and array.
- Able to understand and use typedef.
- Able to understand and use union.
- Able to understand and use enumeration (enum).

11.1 Structure (struct)

- We have learned that by using an array, we only can declare **one data type per array**, and it is same for other data types. To use the same data type with different name, we need another declaration.
- struct data type overcomes this problem by declaring aggregate data types.
- A structure is a collection of related data items stored in one place and can be referenced by more than one names. Usually these data items are different basic data types. Therefore, the number of bytes required to store them may also vary.
- It is very useful construct used in data structure, although in C++, many struct constructs has been replaced by class construct but you will find it a common in the Win32 programming's APIs.
- In order to use a structure, we must first declare a structure template. The variables in a structure are called **elements** or **members**.
- For example, to store and process a student's record with the elements id_num (identification number), name, gender and age, we can declare the following structure.

```
struct student {
    char id_num[5];
    char name[10];
    char gender;
    int age;
    };
```

- Here, struct is a keyword that tells the compiler that a structure template is being declared and student is a **tag** that identifies its data structure. Tag is not a variable; it is **a label** for the structure's template. Note that there is a semicolon after the closing curly brace.
- A structure tags simply a label for the structure's template but you name the structure tag using the same rules for naming variables. The template for the structure can be illustrated as follow (note the different data size):



- Compiler will not reserve memory for a structure until you declare a structure variable same as you would declare normal variables such as int or float.
- Declaring structure variables can be done in any of the following ways (by referring to the previous example):

```
1.    struct  student{
 char id_num[5];
 char name[10];
 char gender;
 int age;
 } studno_1, studno_2;

2.    struct{//no tag
```

```
char id_num[5];
char name[10];
char gender;
int age;
}studno_1, studno_2;
```

```
3.    struct student{
char id_num[5];
char name[10];
char gender;
int age;
};
```

- Based on no. 3 version, then in programs we can declare the structure something like this:

```
struct   student   studno_1, studno_2;
```

- In the above three cases, two structure variables, studno_1 and studno_2, are declared. Each structure variable has 4 elements that is 3 character variables and an integer variable.
- In (1) and (2), the structure variables are declared immediately after the closing brace in the structure declaration whereas in (3) they are declared as student. Also in (2) there is no structure tag, this means we cannot declare structure variables of this type elsewhere in the program instead we have to use the structure variables studno_1 and studno_2 directly.
- The most widely used may be no (1) and (3) where we put the declaration of the struct in header files and use it anywhere in programs as follows:

```
struct   student   studno_1, studno_2;
```

- Where the studno_1, studno_2 are variables declared as usual but the type here is struct student instead of integral type such as int, char and float.
- It is also a normal practice to combine the typedef (will be explained later on) with struct, making the variables declaration even simpler. For example:

```
typedef struct TOKEN_SOURCE {
  CHAR    SourceName[8];
  LUID    SourceIdentifier;
} TOKEN_SOURCE, *PTOKEN_SOURCE;
```

- In this example we use typedef with struct. In our program we just declare variable as follows:

```
TOKEN_SOURCE  myToken;
```

- The TOKEN_SOURCE type is used for a normal variable and the *PTOKEN_SOURCE type is used for a pointer variable. You will find that these are typical constructs in Win32 APIs of Windows. For more information refers to C & Win32 programming tutorials.

11.2 Accessing The Structure Element

- A structure element can be accessed and assigned a value by using the structure variable name, the dot operator (**.**) and the element's name. For example the following statement:

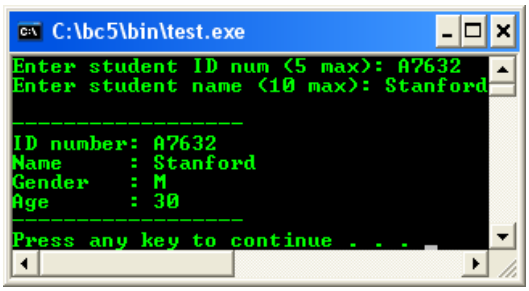
```
studno_1.name = "jasmine";
```

- Assigns string "jasmine" to the element name in the structure variable studno_1. The dot operator simply qualifies that name is an element of the structure variable studno_1. The other structure elements are referenced in a similar way.
- Unfortunately, we cannot assign string "jasmine" (a const char) directly to an array in the structure (char [ ]). For this reason, we have to use other methods such as receiving the string from user input or by using pointers.
- A program example.

```
// a simple structure program example
#include <stdio.h>
#include <stdlib.h>

struct student{
  char id_num[6];
  char name[11];
  char gender;
  int age;
  };

int main(void)
{
```

```
    struct student studno_1;

    // studno_1.id_num = "A3214"; // illegal, const char to char[]
    // studno_1.name = "Smith";   // illegal, const char to char[]
    printf("Enter student ID num (5 max): ");
    scanf("%s", studno_1.id_num);
    printf("Enter student name (10 max): ");
    scanf("%s", studno_1.name);
    studno_1.gender = 'M';
    studno_1.age = 30;

    printf("\n------------------\n");
    printf("ID number: %s\n", studno_1.id_num);
    printf("Name     : %s\n", studno_1.name);
    printf("Gender   : %c\n", studno_1.gender);
    printf("Age      : %d\n", studno_1.age);
    printf("------------------\n");
    return 0;
}
```

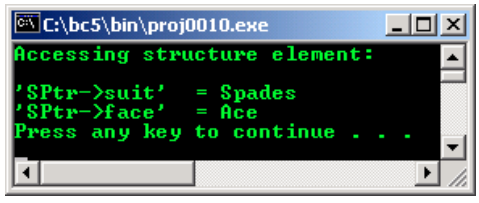
Output:



- The structure pointer operator (→), consisting of a minus (-) sign and a greater than (>) sign with no intervening spaces, accesses a structure member via a pointer to the structure.
- For the following example, by assuming that a pointer SPtr has been declared to point to struct Card, and the address of structure p has been assigned to SPtr. To print member suit of structure Card with pointer SPtr, use the statement Sptr→suit as shown in the following example.

```
// accessing structure element
#include <iostream>
using namespace std;

struct Card
{
  char  *face;  // pointer to char type
  char  *suit;   // pointer to char type
};

void main()
{
   // declare the struct type variables
   struct Card  p;
   struct Card  *SPtr;
   p.face = "Ace";
   p.suit = "Spades";
   SPtr = &p;
   cout<<"Accessing structure element:\n";
   cout<<"\n\'SPtr->suit\'  = "<<SPtr->suit<<endl;
   cout<<"\'SPtr->face\'  = "<<SPtr->face<<endl;
}
```

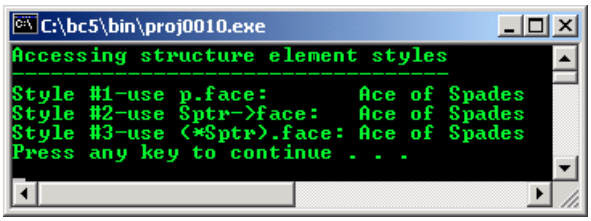
Output:



- The expression SPtr->suit is equivalent to (*SPtr).suit which dereferences the pointer and accesses the member suit using the structure member operator.
- The parentheses are needed here because the structure member operator, the dot (.) has higher precedence than the pointer dereferencing operator, the asterisk (*).
- A program example:

```
// using the structure member and structure
// pointer operators – accessing structure elements styles...
#include <iostream>
using namespace std;

struct Card
{
   char  *face;
   char  *suit;
};

int main()
{
   struct Card  p;
   struct Card  *SPtr;
   p.face = "Ace";
   p.suit = "Spades";
   SPtr = &p;
   cout<<"Accessing structure element styles"<<endl;
   cout<<"----------------------------------"<<endl;
   cout<<"Style #1-use p.face:       "<<p.face<<" of "<<p.suit<<endl;
   cout<<"Style #2-use Sptr->face:   "<<SPtr->face<<" of "<<SPtr->suit<<endl;
   cout<<"Style #3-use (*Sptr).face: "<<(*SPtr).face<<" of "<<(*SPtr).suit<<endl;
   return 0;
}
```

Output:



11.3 Arrays Of Structures

- Suppose you would like to store the information of 100 students. It would be tedious and unproductive to create 100 different student array variables and work with them individually. It would be much easier to create an array of student structures.
- Structures of the same type can be grouped together into an array. We can declare an array of structures just like we would declare a normal array of variables.
- For example, to store and manipulate the information contained in 100 student records, we use the following statement:

```
struct student{
      char id[5];
      char name[80];
      char gender; int age;
      }stud[100];
```

- Or something like the following statements:

```
struct student{
      char id[5];
      char name[80];
      char gender;
      };
```

- And later in our program we can declare something like this:

```
struct   student   stud[100];
```

- These statements declare 100 variables of type struct student (a structure). As in arrays, we can use a subscript to reference a particular student structure or record.
- For example, to print the name of the seventh student, we could write the following statement:

```
cout<<stud[6].name;
```

- Example of initializing all the student names to blanks and their ages to 0, we could do this simply by using for loop as shown below:

```
for(i=0; i<100; i++)
{
  stud[i].name = " ";
  stud[i].age = 0;
}
```

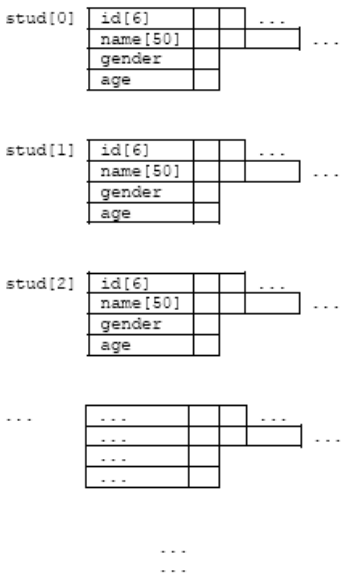
- Very useful huh! Now let try a program example:

```
// an array structure of student information
#include <iostream>
using namespace std;

struct student
{
   char id[6];              // student id number, max. 5 integer number
   char name[50];     // student name, max 49 characters
   char gender;         // student gender Male or Female
   int  age;               // student age
};

void main()
{
   // declaring array of 10 element of structure type
   // and some of the element also are arrays
   struct student stud[10];
   int i = 0;

   cout<<"Keying in student data and then display\n";
   cout<<"----------------------------------------\n";
   cout<<"Enter student data\n";

   for(i=0; i<2; i++)
   {
       // storing the data
       cout<<"\nID number (4 integer number) student #"<<i<<": ";
       cin>>stud[i].id;
       cout<<"First name student #"<<i<<": ";
       cin>>stud[i].name;
       cout<<"Gender (M or F) student #"<<i<<": ";
       cin>>stud[i].gender;
       cout<<"Age student #"<<i<<": ";
       cin>>stud[i].age;
}

   cout<<"\n----------Display the data---------\n";
   cout<<"You can see that the data storage\n";
   cout<<"has been reserved for the structure!\n";
   cout<<"-------------------------------------\n";
   for(i=0; i<2; i++)
   {
   // displaying the stored data
   cout<<"\nID number student  # "<<i<<": "<<stud[i].id;
   cout<<"\nFirst name student # "<<i<<": "<<stud[i].name;
   cout<<"\nGender student     # "<<i<<": "<<stud[i].gender;
   cout<<"\nAge student        # "<<i<<": "<<stud[i].age<<"\n";
   }
}
```

Output:

```
C:\bc5\bin\proj0010.exe
Keying in student data and then display
---------------------------------------
Enter student data

ID number (4 integer number) student #0: 1234
First name student #0: J.Bush
Gender (M or F) student #0: M
Age student #0: 25

ID number (4 integer number) student #1: 9012
First name student #1: Marylane
Gender (M or F) student #1: F
Age student #1: 27

----------Display the data---------
You can see that the data storage
has been reserved for the structure!
---------------------------------------

ID number student  # 0: 1234
First name student # 0: J.Bush
Gender student     # 0: M
Age student        # 0: 25

ID number student  # 1: 9012
First name student # 1: Marylane
Gender student     # 1: F
Age student        # 1: 27
Press any key to continue . . .
```

- The structure template for the program example can be illustrated as follows:



11.4 Structures And Function

- Individual structure elements or even an entire structure can be passed to functions as arguments. For example, to modify the name of the seventh student, let say the function name is modify(), we could issue the following function call:

```
modify(stud[6].name);
```

- This statement passes the structure element name of the seventh student to the function modify(). Only a **copy of name** is passed to the function. This means any change made to name in the called function is local; the value of name in the calling program remains unchanged.
- An entire structure can also be passed to a function. We can do this either by **passing the structure itself** as argument or by simply **passing the address** of the structure.
- For structure element example, to modify the seventh student in the list, we could use any of the following statements.

```
modify(stud[6]);
modify(&stud[6]);
```

- In the first statement, a copy of the structure is passed to the function while in the second only the address of the structure is passed.

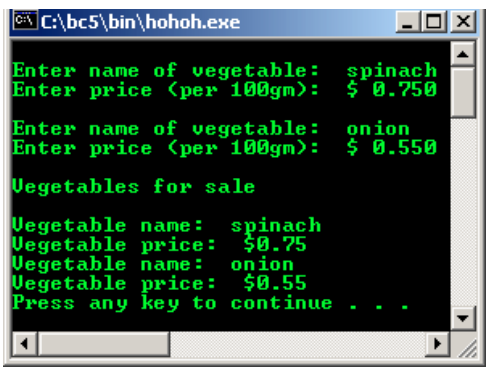
- As only a copy of the structure is passed in the first statement, any changes made to the structure within the called function do not affect the structure in the calling function.
- However in the second statement any changes made to the structure within the called

function will change the structure values in the calling function since the called function directly accesses the structure and its elements.

- Let take a look at a program example.

```
// passing structures to functions and
// a function returning a structure
#include <iostream>
#include <cstdio>
using namespace std;

 // -----structure part---
struct vegetable
{
   char   name[30];
   float  price;
};

// -----main program-----
int main()
{
   // declare 2 structure variables
   struct vegetable veg1, veg2;
   // function prototype of type struct
   struct vegetable  addname();
   // another function prototype
   int list_func(vegetable);

   // functions call for user input...
   veg1  =  addname();
   veg2  =  addname();
   cout<<"\nVegetables for sale\n";

   // function call for data display...
   list_func(veg1);
   list_func(veg2);
   cout<<endl;
   return  0;
}
//------------function----------------
// this functions returns a structure
struct vegetable addname()
{
   char  tmp[20];
   // declare a structure variable
   struct vegetable vege;

   cout<<"\nEnter name of vegetable:  ";
   gets(vege.name);
   cout<<"Enter price (per 100gm):  $ ";
   gets(tmp);
   // converts a string to float
   vege.price = atof(tmp);
   return (vege);
}

// structure passed from main()
int list_func(vegetable list)
{
   cout<<"\nVegetable name:  "<<list.name;
   cout<<"\nVegetable price:  $"<<list.price;
   return 0;
}
```

Output:



- It is also possible to create structures in a structure.

C & C++ programming tutorials

Further related reading and digging:

1. The struct lab worksheets for your practice are: C/C++ struct part 1 and C/C++ struct part 2.
2. Also the combination of the struct, arrays, pointers and function C worksheet 1, C lab worksheet part 2 and C lab worksheet part 3.
3. Check the best selling C/C++ books at Amazon.com.

|< C & C++ Preprocessor Directives | Main | C & C++ Type Specifiers 2 >| C-Extra | Microsoft C/Win32 | Site Index | Download |

C and C++ Type Specifiers: Part 1 | Part 2 |