WEEK 4
STRUCTURES

Dr. Yuzana Win

OUTLINE

- Structures
- Unions
- Typedef
- Enumeration

STRUCTURES

- A structure is a collection of different data types held together in a single unit.
- It is used to represent an entity, which is described with a set of attributes.

- For example,
  An employee is represented by the attributes

  employee code (an Integer number),
  employee's name (a character array),
  Department Code (an Integer number),
  Salary (a Floating point number)
  and so forth.

STRUCTURES

The syntax for declaring a structure is as follows:

```
struct  struct_name
        {
            type variable name, variable name,.....;

            type variable name, variable name,.....;

            type variable name, variable name,.....;

                    :
                    :
            type variable name, variable name,.....;

        };
```

DEFINING THE STRUCTURE

Example:

```
struct  book
  {
      char name;
      float  price;
      int pages;
   };
```

Example

```
struct  book
  {
      char name;
      float price;
      int  pages;
   };
struct book b1;
```

Example

```
struct  book
  {
      char name;
      float price;
      int pages;
   }b1;
```

STRUCTURES

Example:

```
struct  employee
                {
                      int code;
                      char name[20];
                      int dept_code;
                      float salary;
                 } emp1, emp2;
```

- Here, *employee_type* is a Tag. The fields inside this structure are *code*, *name*, *dept_code* and *salary*. The variables of the type *emplyee_type* are *emp1* and *emp2*.

STRUCTURES

- By using the structure tag, other variables, function arguments can be declared as follows

  struct struct-name var1, var2, ...;

- For example, we can define a variables of type employee_type as follows

  struct employee_type emp1, emp2;

ACCESSING STRUCTURE ELEMENTS

- The elements/fields of a structure variable are accessed using dot operator '.'. The syntax for accessing a members of a structure variable is

  variable.field

- For example, to access a fields of a variable emp1, emp2 defined in the above example , we can use the dot operator as follows

```
emp1.code
emp1.name
emp1.dept_code
emp1.salary

emp2.code
emp2.name
emp2.dept_code
emp2.salary
```

EXAMPLE

```
// parts.cpp
// uses parts inventory to demonstrate structures
#include <iostream>
using namespace std;
struct part                    //declare a structure
{
int modelnumber;               //ID number of widget
int partnumber;                //ID number of widget part
float cost;                    //cost of part
};
int main()
{
      part part1;              //define a structure variable
      part1.modelnumber = 6244;  //give values to structure members
      part1.partnumber = 373;
      part1.cost = 217.55F;          //display structure members
      cout << "Model"    << part1.modelnumber;
      cout << ", part"    << part1.partnumber;
      cout << ", costs $" << part1.cost << endl;
      return 0; }
```

Output:

Model 6244, part 373, costs $217.55

INITIALIZING STRUCTURES

- Like simple variables and arrays, structure variables can also be initialized at the time of declaration. The format used is quite similar to initialize an array.

Example :

```
struct  stud_type
      {
        int rollnum;
        char name[20];
        int semester;
        float avg;
       };
struct stud_type  stud1={1001,"Aldina", 1, 90.78},
                  stud2={1002, "Raja", 1,79.28};
```

ARRAY OF STRUCTURES

Example :

```
struct  stud_type
{
        int rollnum;
        char name[20];
        int semester;
        float avg;
};
struct stud_type  stud[50];
```

ARRAY OF STRUCTURES

- In the above example, **stud** is an array that consists of 50 structure of type **stud_type**. To access the 0th structure, we can do it as follows,

  stud[0].rollnum

  stud[0].name

  stud[0].semester

  stud[0].avg

EXAMPLE

```
#include <iostream>
using namespace std;
main()
{
    struct book
    {
        char name[10];
        float price;
        int pages;
    };
    struct book b[3];
    int i, n;
    cout << "Enter no: of Book:\n";
    cin >> n;
    for(i=0; i<n; i++)
    {
        cout << "\nEnter Book information for: \n" << i+1 << endl;
        cout << "Enter name, price and pages: " << endl;
        cin >> b[i].name >> b[i].price >> b[i].pages;
    }
    for(i=0; i<n; i++)
    {
        cout << "\nBook information for: \n" << i+1 << endl;
        cout << "Name: " << b[i].name << endl;
        cout << "Price: " << b[i].price << endl;
        cout << "Pages:" << b[i].pages << endl;
    }
}
```

```
Enter no: of Book:
2

Enter Book information for:
1
Enter name, price and pages:
NLP 35.7 34

Enter Book information for:
2
Enter name, price and pages:
AI 40.4 53

Book information for:
1
Name: NLP
Price: 35.7
Pages:34

Book information for:
2
Name: AI
Price: 40.4
Pages:53
```

POINTERS AND STRUCTURES

- As we can use pointers with other types, we can also use pointers for structure variables. We can declare pointer to a student structure as follows

  struct stud_type *ptr;

- In this declaration, **ptr** is a pointer type variable, which can hold an address of a variable of the type stud_type.

POINTERS AND STRUCTURES

```
struct  stud_type
{
      int rollnum;
      char name[20];
      int semester;
      float avg;
};
```

struct stud_type stud={1001,"Aldina", 1, 98.23}, *ptr ;

- Now to make ptr to point to the structure stud, we can write as

  ptr = &stud;

- After this assignment, the structure **stud** can be accessed through the pointer **ptr**. The notation for referring a field of a structure pointed by a pointer is as follows

  (*ptr).field (OR) ptr -> field

- So, We can access the fields of a **stud** through a pointer **ptr** as follows

  cout << ptr->rollnum << ptr->name << ptr->semester << ptr->avg;

EXAMPLE

```
#include <iostream>
using namespace std;
main()
{
	struct student
	{
		char name[10];
		int rollnum;
		int semester;
		float avg;
	};
	struct student st = {"John", 1,1,35.7}, *ptr;
	ptr = &st;
	cout << ptr-> name << endl;
	cout << ptr-> rollnum << endl;
	cout << ptr-> semester << endl;
	cout << ptr-> avg << endl;
}
```

```
John
1
1
35.7
Press any key to continue . . .
```

NESTED STRUCTURE

- Just as there can be arrays of arrays, there can be a structure that contains other structures. This can be a powerful tool to create complex data types.

Example :

```
struct  date
{
     int day;
     int month;
     int year;
};
```

NESTED STRUCTURE (CON'T)

```
struct employee_type
        {
            int code;
            char name[20];
            struct date doj;
            int dept_code;
            float salary;
        }emp1,emp2;
```

- In this example, if we want to access the year of joining of an employee of **emp1**, then we can do so by writing

  emp1.doj.year

PASSING STRUCTURE TO FUNCTION

1. Passing Specified Member
   - Passing value of specified member
     - fun_name(struct_name . member_name);
       - E.g func(person.age);
   - Passing address of specified member
     - fun_name (&struct_name.member_name)
       - E.g func(&person.age)

Notes:

- If member is a string no need to use & because string element already took a address.

PASSING STRUCTURE TO FUNCTION (CON'T)

- Passing Entire Structure
  - Passing by Value
    - fun_name(struct_name)
      - E.g func(person)
  - Passing by Reference
    - fun_name (&struct_name)
      - E.g func(&person)

Notes:

structure pointer is needed to get structure by called function.

EXAMPLE

```
#include <iostream>
using namespace std;

void display(struct person *p);     //Note the declaration of structer pointer
struct person
{
  char name[30];
  int age;
};

void display(struct person *p)
{
  cout << "Name of person is \n" << p->name <<endl;
  cout << "Age of person is \n" << p->age << endl;
}

int main()
{
  struct person per;
  strcpy(per.name, "Edwin");
  per.age = 20;
  display(&per);
  return 0;
}
```

```
Name of person is
Edwin
Age of person is
20
Press any key to continue . . .
```

NOTES

- The information contained in one structure can be assigned to another structure of the same type using a single assignment statement.
- Do not need to assign the value of each member separately

UNIONS

- As like structures, **Union** is also used to group data fields of different type, but unlike structures all data fields are sharing a common memory.

- That is, a union permits a section of memory to be treated as a variable of one type on one occasion, and as a different variable of a different type on a another occasion.
- Thus space need only be allocated to accommodate the largest type specified in a union.

- The syntax for declaring a union, declaring variables of union type, accessing elements of union is identical to that of structures. Union differs from structure in storage and in initialization.

UNIONS (CON'T)

- Unions are used frequently when specialized type conversions are needed because you can refer to the data held in the **union** in fundamentally different ways.

UNIONS

Example

```
union  u_type
  {
            char  var1;
            int    var2;
            float  var3;
            double var4;
          } u_var;
```

UNION OF STRUCTURES

- There can be structures with in unions and unions with in structures. The following block of code illustrates union of structures.

Example

```
struct employee_type
 {
   int code;
   char name[20];
   int dept_code;
   float salary;
 };
```

UNION OF STRUCTURES (CON'T)

```
struct stud_type
        {
         int rollno;
         char name[20];
         int age;
         float avg;
         };
 union  person
        {
            struct employee_type e1;
            struct stud_type  s1;
        } ex;
```

UNION OF STRUCTURES (CON'T)

- In the above example, the union makes the structure variables e1 and s1 to share a common memory space.

- That is, the user can use either e1 or s1, but not both at the same time. The elements of this union of structures are accessed using dot operator as follows.

ex.e1.name

EXAMPLE

```
#include <iostream>
using namespace std;
struct employee_type
        {
                int code;
                char name[20];
                int dept_code;
                float salary;
};

struct stud_type
        {
                int rollno;
                char name[20];
                int age;
                float avg;
         };
```

```
union  person
        {
          struct employee_type e1;
          struct stud_type s1;
        } ex;
main()
{
      person ex;
      cout <<"Employee name \n";
      cin >> ex.e1.name;
      cout << "The employee name
is " << ex.e1.name << endl ;
}
```

```
Employee name
John
The employee name is John
Press any key to continue . . .
```

TYPEDEF KEYWORD

- Defines a new name for an existing type.
- not actually creating a new data type
- can help make machine-dependent programs more portable.
- General form :typedef *type newname*;

  E.g **typedef float balance;**

  //create a **float** variable using **balance**:

TYPEDEF STATEMENT

- typedef can also be used with structures. The following creates a new type agun which is of type struct gun and can be initialised as usual:

```
typedef struct gun
{
   char name[50];
   int magazinesize;
   float calibre;
} agun;

agun arnies= {"Uzi",30,7};
```

- Here gun still acts as a *tag* to the struct and is optional.

- **agun** is the new data type. **arnies** is a variable of type agun which is a structure.

SUMMARY

- Structures are used to group different types of data, and to represent any real-time entity
- Union is similar to structures but, memory is shared by the members
- typedef is used to define a new data type

ENUMERATIONS

- Enumerated types contain a list of constants that can be addressed in integer values.
- We can declare types and variables as follows.

  enum days {mon, tues, ..., sun} week;

  enum days week1, week2;
- As with arrays first enumerated name has index value 0. So **mon** has value 0, tues 1, and so on.

  week1 and week2 are variables.
- We can also override the 0 start value:

  enum months {jan = 1, feb, mar, ..., dec};

  Here it is implied that feb = 2 *etc*.

EXAMPLE

```
// dayenum.cpp
// demonstrates enum types
#include <iostream>
using namespace std;
//specify enum type
enum days_of_week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
int main()
{
    days_of_week day1, day2;  //define variables of type days_of_week

    day1 = Mon;                                  //give values to
   day2 = Thu;                                    //variables
   int diff = day2 - day1;              //can do integer arithmetic
   cout << “Days between=” << diff << endl;
   if(day1 < day2)                          //can do comparisons
      cout << “day1 comes before day2\n”;
   return 0;
}
```

ASSIGNMENTS

1. A phone number, such as (212) 767-8900, can be thought of as having three parts: the area code (212), the exchange (767), and the number (8900).

Write a program that uses a structure to store these three parts of a phone number separately. Call the structure phone. Create two structure variables of type phone. Initialize one, and have the user input a number for the other one. Then display both numbers. The interchange might look like this:

Enter your area code, exchange, and number: 415 555 1212

My number is (212) 767-8900

Your number is (415) 555-1212

ASSIGNMENTS

2. Write a program that uses a structure called point to model a point. Define three points, and have the user input values to two of them. Then set the third point equal to the sum of the other two, and display the value of the new point. Interaction with the program might look like this:

Enter coordinates for p1: 3 4

Enter coordinates for p2: 5 7

Coordinates of p1+p2 are: 8, 11

ASSIGNMENTS

3. Creates Date structure which contains day, month and year member data. Write a C++ program which accepts member data from keyboard and display them.

4. Creates Book structure which contains title, author, ISBN number and price member data. Write a C++ program that it accepts five books from keyboard and display them by using structure array.

5. Creates Library_book structure which contains a Book structure and a Date structure. Write a C++ program for creating Library_book array for three books and display that array.