

USER DEFINED TYPE

Pemrograman

C++ Dasar



Structure

- A structure is a user defined data type. We know that arrays can be used to represent a group of data items that belong to the same type, such as int or float. However we cannot use an array if we want to represent a collection of data items of different types using a single name. A structure is a convenient tool for handling a group of logically related data items.
- Structure is a user defined data type used to represent a group of data items of different types using a single name.

Structure / Records

A structure is a user defined data type that groups logically related data items of different data types into a single unit.

All the elements of a structure are stored at contiguous memory locations.

A variable of structure type can store multiple data items of different data types under the one name.

- Structures hold data that belong **together**.
- Examples:
 - Student record: student id, name, major, gender, start year, ...
 - Bank account: account number, name, currency, balance, ...
 - Address book: name, address, telephone number, ...
- In database applications, structures are called records.

Structures

- Individual components of a struct type are called **members** (or **fields**) or **attributes**.
- Members can be of **different types** (simple, array or struct).
- A struct is named as a whole while individual members are named using field identifiers.
- Complex data structures can be formed by defining **arrays of structs**.

struct basics



- Definition of a structure:

```
struct <struct_name>{  
    <type> <variable_name>;  
    <type> <variable_name>;  
    ...  
} ;
```



Each variable defines a member of the structure.

- Example:

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



The “Date” structure has 3 members, day, month & year.

- In structure declaration the keyword struct appears first, this followed by structure name. The member of structure should be enclosed between a pair of braces and it defines one by one each ending with a semicolon. It can also be array of structure. There is an enclosing brace at the end of declaration and it end with a semicolon.
- We can declare structure variables as follows

```
struct structure_name var1,var2,.....,var n;
```


Example:

- To store the names, roll number and total mark of a student you can declare 3 variables. To store this data for more than one student 3 separate arrays may be declared. Another choice is to make a structure. No memory is allocated when a structure is declared. It just defines the “form” of the structure. When a variable is made then memory is allocated. This is equivalent to saying that there's no memory for “int”, but when we declare an integer that is `int var`; only then memory is allocated. The structure for the above mentioned case will look like

```
struct student
{
    int rollno;
    char name[25];
    float totalmark;
};
```

We can now declare structure variables stud1, stud2 as follows :

```
struct student stud1,stud2;
```

Thus, the stud1 and stud2 are structure variables of type student. The above structure can hold information of 2 students.

- It is possible to combine the declaration of structure combination with that of the structure variables, as shown below.

```
struct structure_name  
{  
    type element 1;  
    type element 2;  
    .....  
    type element n;  
} var1,var2,...,varn;
```

- The following single declaration is equivalent to the two declarations presented in the previous example.

```
struct student
```

```
{
```

```
    int rollno;
```

```
    char name[25];
```

```
    float totalmark;
```

```
} stud1, stud2;
```


Define, declare and initialize Struct

Define the structure

```
// declare a structure

struct StudentRecord{
    char Name[22];
    int Id;
    char Dept[22];
    char Gender;
};
```

```
// declare a structure

struct StudentRecord{
    char Name[22];
    int Id;
    char Dept[22];
    char Gender;
} newStudent ;
```

Declare the variable of structure

```
struct StudentRecord student1;
```

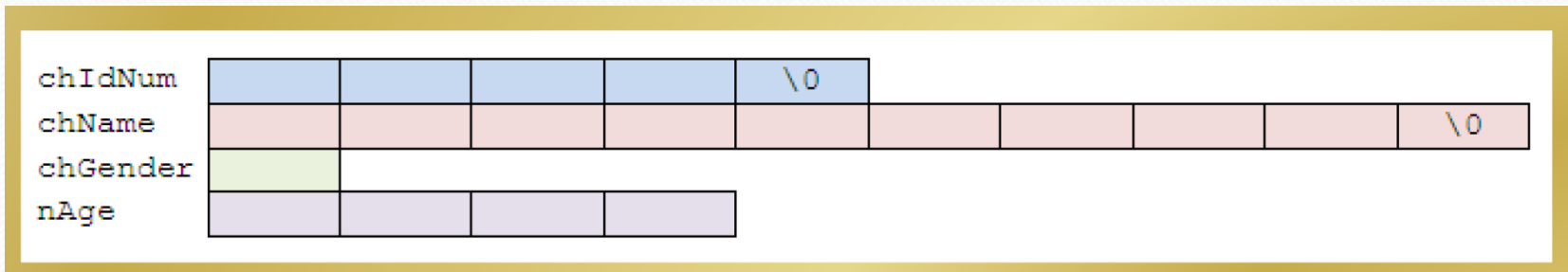
```
newStudent student;
```

Example Struct

To store and process a student's record with the elements chIdNum(identification number), chName, chGender and nAge, We can declare the following structure :

```
struct student{  
    char chIdNum[5];  
    char chName[10];  
    char chGender;  
    int nAge;  
};
```

The previous sample template for the structure can be illustrated as follow (note the different data sizes),



- The struct MyEmployee structure has three members: chName, nldNum, and nDepatment.
- The chName member is a 20-element array, and nldNum and nDepatment are simple members with int and long types, respectively.
- The identifier MyEmployee is the structure identifier.
- Declaring structure variables can be done in the following way,

```
struct MyEmployee {  
    char chName[20];  
    int nldNum;  
    long nDepatment;  
} EmpData;    // defines a structure variable named EmpData
```

- Then we can declare variables of type struct MyEmployee like the following,
- `struct MyEmployee student, staff, academician;`

Accessing structure Variable

- The different variable types stored in a structure are called its members. The structure member can be accessed by using a dot (.) operator, so the dot operator is known as structure member operator

Example:

- In the above example stud1 is a structure variable of type student. To access the member name, we would write stud1.name. Similarly, stud1's rollno and stud1's totalmark can be accessed by writing stud1.rollno and stud1.totalmark respectively.

Accessing struct members

The **member selection operator** (.) is used between the variable name and the member identifier to access individual members of a struct type variable.

EXAMPLES

```
student1.Name;
```

```
student1.Id;
```

```
student1.Dept;
```

```
student1.Gender;
```

Basic Operations on Records

```
student1.Name = "Silmi Fauziati ";  
student1.Id = 1234;  
student1.Dept = "Technology Informaion";  
student1.Gender = 'F';
```

```
printf ("The name of student is : %s", student1.Name);  
printf ("The ID of student is : %d", student1.Id);  
printf ("The Department of student is : %s", student1.Dept);  
printf ("The Gender of student is : %c", student1.Gender);
```



Example of Structure

The structure of Employee is declared as

```
struct employee
{
int emp_id;
char name[20];
float salary;
char address[50];
int dept_no;
int age;
};
```

Memory Space Allocation

8000

8002

8022

8024

8074

8076

8078

emp_id
name[20]
salary
address[50]
dept_no
age

employee

Initializing a Structure Members

The members of individual structure variable is initialize one by one or in a single statement.

The example to initialize a structure variable is

1) struct employee e1 = {1, "Hemant", 12000, "3 vikas colony new delhi", 10, 35};

2) e1.emp_id=1;

e1.dept_no=1

e1.name="Hemant";

e1.age=35;

e1.salary=12000;

e1.address="3 vikas colony new delhi";

Program to implement the Structure

```
#include <stdio.h>
#include <conio.h>
```

```
struct employee
{
    int emp_id;
    char name[20];
    float salary;
    char address[50];
    int dept_no;
    int age;
};
```

```
#include <stdio.h>
#include <conio.h>
```

```
struct employee
{
    int emp_id;
    char name[20];
    float salary;
    char address[50];
    int dept_no;
    int age;
} e1;
```



```
void main ( )
{
    struct employee e1,e2;
    printf ("Enter the id of employee");
    scanf ("%d",&e1.emp_id);
    printf ("Enter the name of employee");
    scanf ("%s",e1.name);
    printf ("Enter the salary of employee");
    scanf ("%f",&e1.salary);
    printf ("Enter the address of employee");
    scanf ("%s",e1.address);
    printf ("Enter the department of
employee");
    scanf ("%d",&e1.dept_no);
    printf ("Enter the age of employee");
```

```
void main ( )
{
    printf ("Enter the id of employee");
    scanf ("%d",&e1.emp_id);
    printf ("Enter the name of employee");
    scanf ("%s",e1.name);
    printf ("Enter the salary of employee");
    scanf ("%f",&e1.salary);
    printf ("Enter the address of employee");
    scanf ("%s",e1.address);
    printf ("Enter the department of
employee");
    scanf ("%d",&e1.dept_no);
    printf ("Enter the age of employee");
```

```
scanf("%d",&e1.age);
printf ("Enter the employee id of employee");
scanf("%d",&e2.emp_id);
printf ("Enter the name of employee");
scanf("%s",e2.name);
printf ("Enter the salary of employee");
scanf("%f",&e2.salary);
printf ("Enter the address of employee");
scanf("%s",&e2.address);
printf ("Enter the department of employee");
scanf("%d",&e2.dept_no);
printf ("Enter the age of employee");
scanf("%d",&e2.age);
```



```
printf ("The employee id of employee is : %d", e1.emp_id);  
printf ("The name of employee is : %s", e1.name);  
printf ("The salary of employee is : %f", e1.salary);  
printf ("The address of employee is : %s", e1.address);  
printf ("The department of employee is : %d", e1.dept_no);  
printf ("The age of employee is : %d", e1.age);
```

```
printf ("The employee id of employee is : %d", e2.emp_id);  
printf ("The name of employee is : %s", e2.name);  
printf ("The salary of employee is : %f", e2.salary);  
printf ("The address of employee is : %s", e2.address);  
printf ("The department of employee is : %d", e2.dept_no);  
printf ("The age of employee is : %d",e2.age);  
getch();  
}
```


Output of Program

Enter the employee id of employee 1

Enter the name of employee Rahul

Enter the salary of employee 15000

Enter the address of employee 4,villa area, Delhi

Enter the department of employee 3

Enter the age of employee 35

Enter the employee id of employee 2

Enter the name of employee Rajeev

Enter the salary of employee 14500

Enter the address of employee flat 56H, Mumbai

Enter the department of employee 5

Enter the age of employee 30



The employee id of employee is : 1

The name of employee is : Rahul

The salary of employee is : 15000

The address of employee is : 4, villa area, Delhi

The department of employee is : 3

The age of employee is : 35

The employee id of employee is : 2

The name of employee is : Rajeev

The salary of employee is : 14500


The address of employee is : flat 56H, Mumbai

The department of employee is : 5

The age of employee is : 30

Union

- Union is a user created data type similar to structure but in this case all the members share a common memory location. The size of the union corresponds to the length of the largest member. Since the member share a common location they have the same starting address.
- The real purpose of unions is to prevent memory fragmentation by arranging for a standard size for data in the memory.

- 
- By having a standard data size we can guarantee that any hole left when dynamically allocated memory is freed will always be reusable by another instance of the same type of union. This is a natural strategy in system programming where many instances of different kinds of variables with a related purpose and stored dynamically.

- A union is declared in the same way as a structure. The syntax of union declaration is `union union_name`
{
 type element 1;
 type element 2;

 type element n;
};

This declares a type template. Variables are then declared as:

```
union union_name x,y,z;
```

- For example, the following code declares a union data type called Student and a union variable called stud:

```
union student
{
    int rollno;
    float totalmark;
};
union student stud;
```


- It is possible to combine the declaration of union combination with that of the union variables, as shown below.

```
union union_name
{
    type element 1;
    type element 2;
    .....
    type element n;
}var1,var2,...,varn;
```

- The following single declaration is equivalent to the two declaration presented in the previous example.

union student

{

int rollno;

float totalmark;

}x,y,z;


```

#include<stdio.h>
#include<conio.h>
void main()
{
    struct testing {
        int a;
        char b;
        float c;
    }var;
    clrscr();
    printf("\nsizeof(var) is %d",sizeof(var));
    printf("\nsizeof(var.a) is %d",sizeof(var.a));
    printf("\nsizeof(var.b) is %d",sizeof(var.b));
    printf("\nsizeof(var.c) is %d",sizeof(var.c));
    var.a=10;
    printf("\nvalue of var.a is %d",var.a);

```

```

var.b='b';
printf("\nvalue of var.b is %c",var.b);
var.c=15.55;
printf("\nvalue of var.c is %f",var.c);
printf("\nvalue of var.a is %d",var.a);
printf("\nvalue of var.b is %c",var.b);
printf("\nvalue of var.c is %f",var.c);
getch();
}

```

OUTPUT

```

sizeof(var) is 7
sizeof(var.a) is 2
sizeof(var.b) is 1
sizeof(var.c) is 4
value of var.a is 10
value of var.b is b
value of var.c is 15.550000
value of var.a is 10
value of var.b is b
value of var.c is 15.550000

```

- Union:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    union testing
    {
        int a;
        char b;
        float c;
    }var;
    clrscr();
    printf("\nsizeof(var) is %d",sizeof(var));
    printf("\nsizeof(var.a) is %d",sizeof(var.a));
    printf("\nsizeof(var.b) is %d",sizeof(var.b));
    printf("\nsizeof(var.c) is %d",sizeof(var.c));
    var.a=10;
```

```
printf("\nvalue of var.a is %d",var.a);
var.b='b';
printf("\nvalue of var.b is %c",var.b);
var.c=15.55;
printf("\nvalue of var.a is %f",var.c);
printf("\nvalue of var.a is %d",var.a);
printf("\nvalue of var.b is %c",var.b);
printf("\nvalue of var.c is %f",var.c);
getch();
}
```

OUTPUT

```
sizeof(var) is 4
sizeof(var.a) is 2
sizeof(var.b) is 1
sizeof(var.c) is 4
value of var.a is 10
value of var.b is b
value of var.c is 15.550000
value of var.a is -13458
value of var.b is
value of var.c is 15.550000
```


Unions

- A union is similar to a structure, except that its members are overlaid (located at the same memory address).

- Example:

```
union {  
    int i;  
    double d;  
} u;
```

- The members of a union are accessed in the same way as members of a structure:

u.i = 15;

or

u.d = 8.89;

Since u.i and u.d have the same memory address, changing the value of one alters the value of the other.

Unions

- Unions have the same properties as structures:

Unions can be initialized (the initializer specifies the value of the first member).

Unions can be identified by tags or type names.

Unions can be assigned, passed to functions, and returned by functions.

The union data type allocate the space equal to space need to hold the largest data member of union

The memory occupied by structure variable is the sum of sizes of all the members but memory occupied by union variable is equal to space hold by the largest data member of a union.


```
1  /* Fig. 10.5: fig10_05.c
2     An example of a union */
3  #include <stdio.h>
4
5  /* number union definition */
6  union number {
7     int x;
8     double y;
9  }; /* end union number */
10
11 int main( void )
12 {
13     union number value; /* define union variable */
14
15     value.x = 100; /* put an integer into the union */
16     printf( "%s\n%s\n%s\n %d\n\n%s\n %f\n\n\n",
17         "Put a value in the integer member",
18         "and print both members.",
19         "int:", value.x,
20         "double:", value.y );
21 }
```

Fig. 10.5 | Displaying the value of a union in both member data types. (Part I of 2.)

```
22 value.y = 100.0; /* put a double into the same union */
23 printf( "%s\n%s\n%s\n %d\n\n%s\n %f\n",
24         "Put a value in the floating member",
25         "and print both members.",
26         "int:", value.x,
27         "double:", value.y );
28 return 0; /* indicates successful termination */
29 }
```

Put a value in the integer member and print both members.

```
int:
```

100

double:

[illegible]

Put a value in the floating member and print both members.

```
int:
```

0

double:

100.000000

Fig. 10.5 | Displaying the value of a union in both member data types. (Part 2 of 2.)

Array of structures

- It is possible to store a structure as an array element. i.e., an array in which each element is a structure. Just as arrays of any basic type of variable are allowed, so are arrays of a given type of structure. Although a structure contains many different types, the compiler never gets to know this information because it is hidden away inside a sealed structure capsule, so it can believe that all the elements in the array have the same type, even though that type is itself made up of lots of different types.

- The declaration statement is given below.

```
struct struct_name  
{  
    type element 1;  
    type element 2;  
    .....  
    type element n;  
}array name[size];
```

Example:

```
struct student  
{  
    int rollno;  
    char name[25];  
    float totalmark;  
} stud[100];
```


- In this declaration stud is a 100-element array of structures. Hence, each element of stud is a separate structure of type student. An array of structure can be assigned initial values just as any other array. So the above structure can hold information of 100 students.

Program to demonstrate use of array of structure

```
#include <stdio.h>
#include <conio.h>
void main()
{
    struct student
    {
        int rollno;
        char name[25];
        int totalmark;
    }stud[100];
```

```

int n,i;
clrscr();
printf("Enter total number of students\n\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter details of %d-th student\n",i+1);
printf("Name:\n");
scanf("%s",&stud[i].name);
printf("Roll number:\n");
scanf("%d",&stud[i].rollno);
printf("Total mark:\n");
scanf("%d",&stud[i].totalmark);
}
printf("STUDENTS DETAILS:\n");
for(i=0;i<n;i++)
{
printf("\nRoll number:%d\n",stud[i].rollno);
printf("Name:%s\n",stud[i].name);
printf("Total mark:%d\n",stud[i].totalmark);
}
getch();
}

```

OUTPUT

```

Enter total number of students:
3
Enter details of 1-th student
Name:SUBAHAS
Roll number:11
Total mark:589
Enter details of 2-th student
Name:RUKSANA
Roll number:12
Total mark:594
Enter details of 3-th student
Name:SANA
Roll number:13
Total mark:595
STUDENTS DETAILS:
Roll number:11
Name: SUBAHAS
Total mark:589
Roll number:12
Name: RUKSANA
Total mark:594
Roll number:13
Name: SANA
Total mark:595

```


STRUCT & UNION USING C++

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

```
struct Person
{
    string first_name;
    string last_name;
    int age;   float salary;
};
```

```
int main()
{
    Person p1;
    cout << "Enter first name: ";
    cin >> p1.first_name;
    cout << "Enter last name: ";
    cin >> p1.last_name;
    cout << "Enter age: ";
    cin >> p1.age;
```

```
    cout << "Enter salary: ";
    cin >> p1.salary;
```

```
    cout << "\nDisplaying Information." << endl;
    cout << "First Name: " << p1.first_name << endl;
    cout << "Last Name: " << p1.last_name << endl;
    cout << "Age: " << p1.age << endl;
    cout << "Salary: " << p1.salary;
    return 0;
}
```

Output

```
Enter first name: Jane
Enter last name: Smith
Enter age: 27
Enter salary: 10000
```

```
Displaying Information.
First Name: Jane
Last Name: Smith
Age: 27
Salary: 10000
```

```
#include <iostream>
using namespace std;
union Data {
    int intValue;
    float floatValue;
    char charValue;
};
int main() {
    Data data;
    // Assigning an integer value to the union
    data.intValue = 10;
    cout << "Integer value: " << data.intValue << endl;
    // Assigning a float value to the union
    data.floatValue = 3.14f;
    cout << "Float value: " << data.floatValue << endl; // Note: This will overwrite the intValue
    // Assigning a character value to the union
    data.charValue = 'A';
    cout << "Character value: " << data.charValue << endl; // Note: This will overwrite the floatValue
    return 0;
}
```



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

```
// Define a union with different data types
union Student {
    int rollNo;
    float height;
    char firstLetter;
};
```

```
int main()
{
    // Declare a union variable
    Student data;

    data.rollNo = 21;
    cout << data.rollNo << endl;

    data.height = 5.2;
    cout << data.height << endl;

    data.firstLetter = 'N';
    cout << data.firstLetter << endl;
    return 0;
}
```

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

union A {
    int x;
    char y;
};

union B {
    int arr[10];
    char y;
};

int main()
{
    // Finding size using sizeof() operator
    cout << "Sizeof A: " << sizeof(A) << endl;
    cout << "Sizeof B: " << sizeof(B) << endl;

    return 0;
}
```

Output

Sizeof A: 4

Sizeof B: 40

Questions?

