Structure

Introduction

Array of Structure

Pointer to Structure

Nested Structure

Passing Structure to Function

Introduction to Structure

- Problem:
  - How to group together a collection of data items of different types that are logically related to a particular entity??? (~~**Array**~~)

Solution: ***Structure***

Structure

- A structure is a collection of variables of different data types under a single name.
- The variables are called members of the structure.
- The structure is also called a user-defined data type.

Defining a Structure

- Syntax:

```
struct structure_name
        {
        data_type member_variable1;
        data_type member_variable2;
        ………………………………;
        data_type member_variableN;
        };
```

Once structure_name is declared as new data type, then variables of that type can be declared as:

```
        struct structure_name structure_variable;
```

Note: The members of a structure do not occupy memory until they are associated with a structure_variable.

- Example

```
struct student
        {
        char name[20];
        int roll_no;
        float marks;
        char gender;
        long int phone_no;
        };
struct student st;
```

- Multiple variables of *struct student* type can be declared as:

```
struct student st1, st2, st3;
```

Defining a structure…

- Each variable of structure has its own copy of member variables.
- The member variables are accessed using the dot (.) operator or member operator.
- For example: *st1.name* is member variable *name* of *st1* structure variable while *st3.gender* is member variable *gender* of *st3* structure variable.

Defining a structure…

- The structure definition and variable declaration can be combined as:

```
struct student
          {
          char name[20];
          int roll_no;
          float marks;
          char gender;
          long int phone_no;
          }st1, st2, st3;
```

The use of *structure_name* is optional.

```
struct
{
char name[20];
int roll_no;
float marks;
char gender;
long int phone_no;
}st1, st2, st3;
```

Structure initialization

- Syntax:

struct structure_name structure_variable={value1, value2, … , valueN};

- There is a one-to-one correspondence between the members and their initializing values.
- Note: C does not allow the initialization of individual structure members within the structure definition template.

```
struct student
    {
    char name[20];
    int roll_no;
    float marks;
    char gender;
    long int phone_no;
    };
void main()
{
struct  student  st1={“ABC",  4,  79.5,  'M',  5010670};
clrscr();
printf("Name\t\t\tRoll No.\tMarks\t\tGender\tPhone No.");
printf("\n..................................................................................\n");
printf("\n %s\t\t %d\t\t %f\t %c\t %ld",  st1.name, st1.roll_no, st1.marks,
    st1.gender, st1.phone_no);
getch();
}
```

Initialization

Partial Initialization

- We can initialize the first few members and leave the remaining blank.
- However, the uninitialized members should be only at the end of the list.
- The uninitialized members are assigned default values as follows:
  - Zero for integer and floating point numbers.
  - ‘\0’ for characters and strings.

```
struct student
    {
    char name[20];
    int roll;
    char remarks;
    float marks;
    };
void main()
{
struct student s1={“name", 4};
clrscr();
printf("Name=%s", s1.name);
printf("\n Roll=%d", s1.roll);
printf("\n Remarks=%c", s1.remarks);
printf("\n Marks=%f", s1.marks);
getch();
}
```

Accessing member of structure/ Processing a structure

- By using dot (.) operator or period operator or member operator.
- Syntax:

  structure_variable.member

- Here, *structure_variable* refers to the name of a *struct* type variable and *member* refers to the name of a member within the structure.

Question

- Create a structure named *student* that has *name*, *roll and mark* as members. Assume appropriate types and size of member. Write a program using structure to read and display the data entered by the user.

```
struct student
                {
                char name[20];
                int roll;
                float mark;
                };

void main()
{
struct student s;
clrscr();
printf("Enter name:\t");
gets(s.name);
printf("\n Enter roll:\t");
scanf("%d", &s.roll);
printf("\n Enter marks:\t");
scanf("%f", &s.mark);
printf("\n Name \t Roll \t Mark\n");
printf("\n.......................................\n");
printf("\n%s\t%d\t%f", s.name, s.roll, s.mark);
getch();
}
```

Copying and Comparing Structure Variables

- Two variables of the same structure type can be copied in the same way as ordinary variables.
- If *student1* and *student2* belong to the same structure, then the following statements are valid:

  student1=student2;

  student2=student1;

- However, the statements such as:

  student1==student2

  student1!=student2

  are not permitted.
- If we need to compare the structure variables, we may do so by comparing members individually.

```
struct student
    {
    char name[20];
    int roll;
    };
void main()
{
struct student student1={“ABC", 4, };
struct student student2;
clrscr();
student2=student1;
printf("\nStudent2.name=%s", student2.name);
printf("\nStudent2.roll=%d", student2.roll);
if(strcmp(student1.name,student2.name)==0 &&
    (student1.roll==student2.roll))
    {
    printf("\n\n student1 and student2 are same.");
    }
getch();
}
```

Here, structure has been declared global i.e. outside of main() function. Now, any function can access it and create a structure variable.

How structure elements are stored?

- The elements of a structure are always stored in contiguous memory locations.
- A structure variable reserves number of bytes equal to sum of bytes needed to each of its members.
- Computer stores structures using the concept of “word boundary”. In a computer with two bytes word boundary, the structure variables are stored left aligned and consecutively one after the other (with at most one byte unoccupied in between them called slack byte).

How structure elements are stored?

- When we declare structure variables, each one of them may contain slack bytes and the values stored in such slack bytes are undefined.
- Due to this, even if the members of two variables are equal, their structures do not necessarily compare.
- That’s why C does not permit comparison of structures.

Array of structure

- Let us consider we have a structure as:

```
struct student
{
char name[20];
int roll;
char remarks;
float marks;
};
```

- If we want to keep record of 100 students, we have to make 100 structure variables like st1, st2, …,st100.
- In this situation we can use array of structure to store the records of 100 students which is easier and efficient to handle (because loops can be used).

Array of structure…

- Two ways to declare an array of structure:

```
struct student
        {
        char name[20];
        int roll;
        char remarks;
        float marks;
        }st[100];
```

```
  struct student
          {
          char name[20];
          int roll;
          char remarks;
          float marks;
          };
struct student st[100];
```

- Write a program that takes roll_no, fname lname of 5 students and prints the same records in ascending order on the basis of roll_no

Reading values

```
for(i=0; i<5; i++)
    {
        printf("\n Enter roll number:");
        scanf("%d", &s[i].roll_no);

        printf("\n Enter first name:");
        scanf("%s", &s[i].f_name);

        printf("\n Enter Last name:");
        scanf("%s", &s[i].l_name);
     }
```

Sorting values

```
for(i=0; i<5; i++)
   {
         for(j=i+1; j<5; j++)
         {
                  if(s[i].roll_no<s[j].roll_no)
                  {
                      temp = s[i].roll_no;
                      s[i].roll_no=s[j].roll_no;
                      s[j].roll_no=temp;
                  }
          }
      }
```

Question

- Define a structure of employee having data members name, address, age and salary. Take the data for n employees in an array and find the average salary.
- Write a program to read the *name*, *address*, and *salary* of 5 employees using array of structure. Display information of each employee in alphabetical order of their name.

Array within Structure

- We can use single or multi dimensional arrays of type *int* or *float*.
- E.g.

```
          struct student
          {
          char name[20];
          int roll;
          float marks[6];
          };
struct student s[100];
```

Array within structure…

- Here, the member *marks* contains six elements, *marks[0]*, *marks[1]*, …, *marks[5]* indicating marks obtained in six different subjects.
- These elements can be accessed using appropriate subscripts.
- For example, *s[25].marks[3]* refers to the marks obtained in the fourth subject by the 26th student.

Reading Values

```
for(i=0;i<n;i++)
{
        printf("\n Enter information about student%d",i+1);
        printf("\n Name:\t");
        scanf(" %s",  s[i].name);
        printf("\n Class:\t");
        scanf("%d", &s[i]._class);
        printf("\n Section:");
        scanf(" %c", &s[i].section);
        printf("\n Input marks of 6 subjects:\t");
        for(j=0;j<6;j++)
        {
                scanf("%f", &temp);
                s[i].marks[j]=temp;
        }
}
```

Structure within another Structure (Nested Structure)

- Let us consider a structure *personal_record* to store the information of a person as:

-
  ```
  struct personal_record
                  {
                  char name[20];
                  int day_of_birth;
                  int month_of_birth;
                  int year_of_birth;
                  float salary;
                  }person;
  ```

Structure within another Structure (Nested Structure)…

- In the structure above, we can group all the items related to birthday together and declare them under a substructure as:

```
        struct Date
               {
               int day_of_birth;
               int month_of_birth;
               int year_of_birth;
               };
struct personal_record
       {
       char name[20];
       struct Date birthday;
       float salary;
       }person;
```

Structure within another Structure (Nested Structure)…

- Here, the structure *personal_record* contains a member named *birthday* which itself is a structure with 3 members. This is called structure within structure.
- The members contained within the inner structure can be accessed as:

  person.birthday.day_of_birth

  person.birthday.month_of_birth

  person.birthday. year_of_birth

- The other members within the structure personal_record are accessed as usual:

  person.name

  person.salary

```
printf("Enter name:\t");
scanf("%s", person.name);
printf("\nEnter day of birthday:\t");
scanf("%d", &person.birthday.day_of_birth);
printf("\nEnter month of birthday:\t");
scanf("%d", &person.birthday.month_of_birth);
printf("\nEnter year of birthday:\t");
scanf("%d", &person.birthday.year_of_birth);
printf("\nEnter salary:\t");
scanf("%f", &person.salary);
```

Structure within another Structure (Nested Structure)…

- ***Note:- More than one type of structures can be nested…***

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

struct name
    {
    char first_name[10];
    char middle_name[10];
    char last_name[10];
    };

struct personal_record
    {
    float salary;
    struct date birthday,deathday;
    struct name full_name;
    };
```

Assignment

- Create a structure named *date* that has *day*, *month* and *year* as its members. Include this structure as a member in another structure named *employee* which has *name*, *id* and *salary* as other members. Use this structure to read and display employee’s name, id, date of birthday and salary.

Pointer to Structure

- A structure type pointer variable can be declared as:

```
struct book
        {
        char name[20];
        int pages;
        float price;
        };
struct book *bptr;
```

- However, this declaration for a pointer to structure does not allocate any memory for a structure but allocates only for a pointer, so that to access structure's members through pointer ***bptr***, we must allocate the memory using ***malloc()*** function.
- Now, individual structure members are accessed as:

```
bptr->name          bptr->pages          bptr->price
```

OR

```
(*bptr).name        (*bptr).pages        (*bptr).price
```

- Here, **->** is called arrow operator and there must be a pointer to the structure on the left side of this operator.

```
struct book *bptr;

bptr=(struct book *)malloc(sizeof(struct book));

printf("\n Enter name:\t");
scanf("%s", bptr->name);
printf("\n Enter no. of pages:\t");
scanf("%d", &bptr->pages);
printf("\n Enter price:\t");
scanf("%f", & bptr->price=temp)
```

Pointer to Structure…

- Also, the address of a structure type variable can be stored in a structure type pointer variable as follows:

```
struct book
        {
        char name[20];
        int pages;
        float price;
        };
struct book b, *bptr;
bptr=&b;
```

- Here, the base address of *b* is assigned to *bptr* pointer.

Pointer to Structure…

- Now the members of the structure book can be accessed in 3 ways as:

b.name	*bptr->name*	*(*bptr).name*
b.pages	*bptr->pages*	*(*bptr).pages*
b. price	*bptr-> price*	*(*bptr).price*

Pointer to array of structure

- Let we have a structure as follows:

```
    struct book
          {
          char name[20];
          int pages;
          float price;
          };
   struct book b[10], *bptr;
```

- Then the assignment statement *bptr=b;* assigns the address of the zeroth element of *b* to *bptr.*

Pointer to array of structure…

- The members of *b[0]* can be accessed as:

bptr->name   *bptr->pages*   *bptr->price*

- Similarly members of *b[1]* can be accessed as:

(bptr+1)->name   *(bptr+1)->pages*   *(bptr+1)->price*

- The following *for* statement can be used to print all the values of array of structure *b* as:

  for(bptr=b;bptr<b+10;bptr++)

  printf("%s %d %f", bptr->name, bptr->pages, bptr->price);

Problem

- Define a structure of employee having data members name, address, age and salary. Take data for n employee in an array **dynamically** and find the average salary.
- Define a structure of student having data members name, address, marks in C language, and marks in information system. Take data for n students in an array dynamically and find the total marks obtained.

Function and Structure

- We will consider four cases here:
  - *Passing the individual members to functions*
  - *Passing whole structure to functions*
  - *Passing structure pointer to functions*
  - *Passing array of structure to functions*

Passing structure member to functions

- Structure members can be passed to functions as actual arguments in function call like ordinary variables.
- Problem: Huge number of structure members
- Example: Let us consider a structure *employee* having members *name*, *id* and *salary* and pass these members to a function:

```
display(emp.name,emp.id,emp.salary);
```

```
Void display(char e[],int id ,float sal )
{
printf("\nName\t\tID\t\tSalary\n);
printf("%s\t%d\t%.2f",e,id,sal);
}
```

Passing whole structure to functions

- Whole structure can be passed to a function by the syntax:

```
function_name(structure_variable_name);
```

- The called function has the form:

```
return_type function_name(struct tag_name structure_variable_name)
        {
        … … … … …;
        }
```

```
display(emp);
```

```
void display(struct employee e)
{
printf("\nName\tID\tSalary\n");
printf("%s\t%d\t%.2f",e.name,e.id,e.salar);
}
```

Passing structure pointer to functions

- In this case, address of structure variable is passed as an actual argument to a function.
- The corresponding formal argument must be a structure type pointer variable.
- Note: Any changes made to the members in the called function are directly reflected in the calling function.

```
display(&emp);
```

```
void display(struct employee *e)
{
printf("\nName\tID\tSalary\n");
printf("%s\t%d\t%.2f",e->name,e->id,e->salary);
}
```

Passing array of structures to function

- Passing an array of structure type to a function is similar to passing an array of any type to a function.
- That is, the name of the array of structure is passed by the calling function which is the base address of the array of structure.
- Note: The function prototype comes after the structure definition.

```
display(emp);  // emp is array name of size 2
```

```
void display(struct employee ee[])
{
int i;
printf("\n Name\t\t ID\t\t Salary\n");
for(i=0;i<2;i++)
    {
    printf("%s\t\t%d\t\t%.2f\n",ee[i].name,ee[i].id,ee[i].salary);
    }
}
```