Lab 11: Structures

Structures—sometimes referred to as aggregates—are collections of related variables under one name. Structures may contain variables of many different data types, in contrast to arrays that contain only elements of the same data type. Structures are commonly used to define records to be stored in files. Pointers and structures facilitate the formation of more complex data structures such as linked lists, queues, stacks and trees

Structures Definitions

Structures are derived data types; they are constructed using objects of other types. Consider the following structure definition:

```
struct card {
char *face;
char *suit;
 };
```

Keyword struct introduces the structure definition. The identifier card is the structure tag, which names the structure definition and is used with the keyword struct to declare variables of the structure type. In this example, the structure type is struct card. Variables declared within the braces of the structure definition are the structure's members. The definition of struct card contains members face and suit of type char *. Structure members can be variables of the primitive data types (e.g., int, float, etc.), or aggregates, such as arrays and other structures

Self-Referential Structures

A structure cannot contain an instance of itself. For example, a variable of type struct employee cannot be declared in the definition for struct employee. A pointer to struct employee, however, may be included. For example,

```
struct employee2
{
char firstName[ 20 ];
char lastName[ 20 ];
int age;
char gender;
 double hourlySalary;
struct employee2 person; /* ERROR */
struct employee2 *ePtr; /* pointer */
 };
```

Defining Variables of Structure

Structure definitions do not reserve any space in memory; rather, each definition creates a new data type that is used to define variables. Structure variables are defined like variables of other types. The definition

```
struct card {
char   *face;
char *suit;
} aCard, deck[ 52 ], *cardPtr;
```

Operations That Can Be Performed

The only valid operations that may be performed on structures are the following: assigning structure variables to structure variables of the same type, taking the address (&) of a structure variable, accessing the members of a structure variable and using the sizeof operator to determine the size of a structure variable.

Initializing Structures

Structures can be initialized using initializer lists as with arrays. To initialize a structure, follow the variable name in the definition with an equals sign and a brace-enclosed, comma-separated list of initializers. For example, the declaration

```
struct card acard={“Three”, “Hearts”};
```

creates variable aCard to be of type struct card and initializes member face to "Three" and

member suit to "Hearts". If there are fewer initializers in the list than members in the structure, the remaining members are automatically initialized to 0 (or NULL if the member is a pointer). Structure variables defined outside a function definition (i.e., externally) are initialized to 0 or NULL if they are not explicitly initialized in the external definition. Structure variables may also be initialized in assignment statements by assigning a structure variable of the same type, or by assigning values to the individual members of the structure.

Accessing Structure Members

Two operators are used to access members of structures: the structure member operator (.) , also called the dot operator, and the structure pointer operator (->)—also called the arrow operator. The structure member operator accesses a structure member via the structure variable name. For example, to print member suit of structure variable aCard, it uses the following statement.

```
printf( "%s", aCard.suit ); /* displays Hearts */
```

```
printf( "%s", cardPtr->suit); /* displays Hearts */
```

The expression cardPtr->suit is equivalent to (*cardPtr).suit, which dereferences the pointer and accesses the member suit using the structure member operator.

Example

```
#include <stdio.h>
int main(void)
 { /* Structure declaration */
 struct horse
 {
 int age;
 int height;
 char  name[20];
 char father[20];
 char mother[20]; };
  struct horse My_first_horse;        /* Structure variable declaration */
  /* Initialize the structure variable from input data */
  printf("Enter the name of the horse: " );
  scanf("%s", My_first_horse.name );          /* Read the horse's name */
  printf("How old is %s? ", My_first_horse.name );
  scanf("%d", &My_first_horse.age );          /* Read the horse's age    */
  printf("How high is %s ( in hands )? ", My_first_horse.name );
  scanf("%d", &My_first_horse.height );       /* Read the horse's height */
  printf("Who is %s's father? ", My_first_horse.name );
  scanf("%s", My_first_horse.father );        /* Get the father's name */
  printf("Who is %s's mother? ", My_first_horse.name );
  scanf("%s", My_first_horse.mother );        /* Get the mother's name */
  /* Now tell them what we know */
   printf("\n%s is %d years old, %d hands high,",My_first_horse.name,
My_first_horse.age, My_first_horse.height);
   printf(" and has %s and %s as parents.\n", y_first_horse.father,My_first_horse.mother );
   return 0;    }
```

Output

```
Enter the name of the horse: Neddy

How old is Neddy? 12

How high is Neddy ( in hands )? 14

Who is Neddy's father? Bertie

Who is Neddy's mother? Nellie

Neddy is 12 years old, 14 hands high, and has Bertie and Nellie as parents.
```

Arrays of Structures

```
#include  <stdio.h>
#include <ctype.h>
#include<string.h>
int main(void) {
struct horse                    /* Structure declaration */
{
          int age;
          int height;
          char    name[20];
          char   father[20];
          char mother[20];
};
 struct horse My_horses[50];        /* Structure array declaration */
 int hcount = 0;                    /* Count of the number of horses */
 char test = '\0';                  /* Test value for ending         */
 for(hcount = 0; hcount<50 ; hcount++ )
 {
          printf("\nDo you want to enter details of a%s horse (Y or N)? ", hcount?"nother " : "" );
          scanf(" %c", &test );
          if(tolower(test) == 'n')
                    break;
          printf("\nEnter the name of the horse: " );
          scanf("%s", My_horses[hcount].name ); /* Read the horse's name */
          printf("\nHow old is %s? ", My_horses[hcount].name );
          scanf("%d", &My_horses[hcount].age ); /* Read the horse's age */
          printf("\nHow high is %s ( in hands )? ", My_horses[hcount].name );
          scanf("%d", &My_horses[hcount].height );
          printf("\nWho is %s's father? ", My_horses[hcount].name );     /* Get the father's name */
          scanf("%s", My_horses[hcount].father );
          printf("\nWho is %s's mother? ", My_horses[hcount].name ); /* Get the mother's name */
          scanf("%s", My_horses[hcount].mother ); }
          /* Now tell them what we know. */
 for(int i = 0 ; i<hcount ; i++ )
 {
 printf("\n\n%s is %d years old, %d hands high,",My_horses[i].name, My_horses[i].age,
My_horses[i].height);
 printf(" and has %s and %s as parents.", My_horses[i].father,My_horses[i].mother );
 }
 return 0; }
```

Pointer of Structures

```
#include  <stdio.h>
#include  <ctype.h>
#include <stdlib.h>         /* For malloc() */
int main(void)
{
      struct horse            /* Structure declaration */
      {
           int age;
           int height;
           char name[20];
           char father[20];
           char mother[20];
        };
 struct horse *phorse[50]; /* pointer to structure array declaration */
 int hcount = 0;          /* Count of the number of horses          */
 char test = '\0';        /* Test value for ending input            */
 for(hcount = 0; hcount < 50 ; hcount++ )
  {
           printf("\nDo you want to enter details of a%s horse (Y or N)? ", hcount?"nother " : "" );
           scanf(" %c", &test );
            if(tolower(test) == 'n')
                    break;
            /* allocate memory to hold a structure     */
           phorse[hcount] = (struct horse*) malloc(sizeof(struct horse));
           printf("\nEnter the name of the horse: " );
           scanf("%s", phorse[hcount]->name ); /* Read the horse's name */
           printf("\nHow old is %s? ", phorse[hcount]->name );
           scanf("%d", &phorse[hcount]->age ); /* Read the horse's age */
           printf("\nHow high is %s ( in hands )? ", phorse[hcount]->name );
           scanf("%d", &phorse[hcount]->height ); /* Read the horse's height */
           printf("\nWho is %s's father? ", phorse[hcount]->name  );
           scanf("%s", phorse[hcount]->father ); /* Get the father's name */
          printf("\nWho is %s's mother? ", phorse[hcount]->name );
          scanf("%s", phorse[hcount]->mother ); /* Get the mother's name */
   }
```

```
/* Now tell them what we know. */
for(int i = 0 ; i < hcount ; i++ )
{
    printf("\n\n%s is %d years old, %d hands high,",phorse[i]->name, phorse[i]->age,
    phorse[i]->height);
    printf(" and has %s and %s as parents.",phorse[i]->father, phorse[i]->mother);
    free(phorse[i]);
}
return 0;
}
```

Using Structures with Functions

Structures may be passed to functions by passing individual structure members, by passing an entire structure or by passing a pointer to a structure. When structures or individual structure members are passed to a function, they are passed by value. Therefore, the members of a caller's structure cannot be modified by the called function. To pass a structure by reference, pass the address of the structure variable. Arrays of structures—like all other arrays—are automatically passed by reference.

Example

```
#include<stdio.h>
struct student
    {
    char name[30];
    float marks;
    };
 void print_student (struct student student2)
{
       printf( "\nYour name is %s\n", student2.name);
       printf( "\nYour Marks are%.f\n", student2.marks);
}
 void read_student_p(struct student student2)
{
       printf( "Enter your name : ");
       gets(student2.name);
       printf( "\nEnter your marks :");
       scanf("%f",&student2.marks);
       print_student(student2);
}
int main ()
{
   struct student student1;
   read_student_p(student1);
}
```

Returning A Pointer To A Structure

```
#include<stdio.h>
#include<stdlib.h>
struct Date
{
        int day;
        int month;
        int year;
};
struct Family     //Family structure Declaration
{
   struct Date dob;
   char name[20];
   char father[20];
   char mother[20];
};
//Function to input data on Family members
Family* f(int count)
{
   Family *ret;// Define structure Pointer
   //Allocate memory for a structure
   ret= (struct Family*)malloc(count*sizeof(struct Family));
   if( !ret)
      return NULL;

   printf("\n----------Deatils of %d persons---------",count);
   for( int i = 0; i < count; ++i)
   {
           printf("\n\n");
           printf("Enter the Person Name: ");    //Read's Name
           gets(ret[i].name);
           printf("The Date of birth of %s is: ", ret[i].name);
           scanf("%d %d %d",&ret[i].dob.day, &ret[i].dob.month, &ret[i].dob.year); //Read's DOB
           fflush(stdin); //Flush so that it doesn't skip the next line
           printf("The name of %s's Father is: ", ret[i].name); //Get's Father Name
           gets(ret[i].father);
           printf("The name of %s's Mother is: ", ret[i].name); //Get's Mother Name
           gets(ret[i].mother);
}
 return ret; //Return address of Family structure
}
```

```
int main() {
    Family *p = f(3); //Function Calling
    if( p) {
            printf("\n--------The BioData of Perons--------");
        for (int i=0;i<3;i++)
        {
            //Now tell what we know
            //Output Family Data
            printf("\n");
            printf("The name of %d person is %s\n",i+1,p[i].name);
            printf("The Father's name of %s is %s\n",p[i].name,p[i].father);
            printf("The Mother's name of %s is %s\n",p[i].name,p[i].mother);
            printf("The Date of Birth of %s is %d %d %d\n",p[i].name,p[i].dob.day,
            p[i].dob.month,p[i].dob.year);
        }
        free( p); // don't forget
    }

    return 0;
}
```

LAB ACTIVITY

1. Define a struct type with the name Length that represents a length in yards, feet, and inches. Define an add() function that will add two Length arguments and return the sum as type Length. Define a second function, show(), that will display the value of its Length argument. Write a program that will use the Length type and the add() and show() functions to sum an arbitrary number of lengths in yards, feet, and inches that are entered from the keyboard and output the total length.

2. Define a struct type that contains a person's name consisting of a first name and a second name, plus the person's phone number. Use this struct in a program that will allow one or more names and corresponding numbers to be entered and will store the entries in an array of structures. The program should allow a second name to be entered and output all the numbers corresponding to the name, and optionally output all the names with their corresponding numbers.

3. Write a C program for billing system of MovInPeak restaurant. The program should perform following tasks: x Show menu to customer for orders.

- Allow the customer to select more than one item from the menu.
- Calculate and print the bill.

Assume that the restaurant offers the following items (the price of each item is shown to the right of the item):

- Omlet $1.45
- French
- Omlet $2.45
- Muffin $0.99
- French Toast $1.99
- Fruit Basket $2.49
- Cereal $0.69
- Coffee $0.75
- Tea $0.50

Notes:

Define a struct MovinPeakMenu that contains following data members.

- menuItem [ ] of type string
- menuPrice [ ] of type double
- Your program must contain following functions:
- Function getData: This function store the food items into the array menuList.
- Function showMenu: This function shows the different items offered by the restaurant and tells the user how to select the items.
- Function printCheck: This function calculates and prints the check.
- (Note that the billing amount should include a 5% tax.).