COMPUTER PROGRAMMING - I

Structures

- We need entities that are collection of dissimilar data types
- For example, suppose you want to store data about a book. You might want to store its name (a string), its price (a float) and number of pages in it (an int).
- If data about say 3 such books is to be stored, then we can follow two approaches:
- Construct individual arrays, one for storing names, another for storing prices and still another for storing number of pages.
- Use a structure variable.

Structures

- A structure contains a number of data types grouped together. These data types may or may not be of the same type.
- Declaring a structure
- struct book
- {
- char name[10] ;
- float price ;
- int pages ;
- } ;

Structures

- Once the new structure data type has been defined one or more variables can be declared to be of that type.
- For example the variables **b1**, **b2**, **b3** can be declared to be of the type **struct book**, as,
- struct book b1, b2, b3 ;

Structures

- To refer to **pages** of the structure defined in our sample program we have to use,
- b1.pages
- Similarly, to refer to **price** we would use,
- b1.price
- Note that before the dot there must always be a structure variable and after the dot there must always be a structure element.

Structures

- The closing brace in the structure type declaration must be followed by a semicolon.
- It is important to understand that a structure type declaration does not tell the compiler to reserve any space in memory. All a structure declaration does is, it defines the ‘form’ of the structure.
- Usually structure type declaration appears at the top of the source code file, before any variables or functions are defined. In very large programs they are usually put in a separate header file, and the file is included (using the preprocessor directive #include) in whichever program we want to use this structure type.

```
struct college
{

    int id ;
    char name[10] ;
    float marks;

} c ;
```

Accessing members of structures

```
int main()
{
    printf("\nEnter the college id : ") ;
        scanf("%d", &c.id ) ;
        printf("\nEnter the name : ") ;
        fflush(stdin);
        gets(c.name);
        printf("\nEnter the marks : ") ;
        scanf("%f", &c.marks) ;

}
```

Arrays of structures

- In our sample program, to store data of 100 students we would be required to use 100 different structure variables from **c1** to **c100**, which is definitely not very convenient. A better approach would be to use an array of structures.

- Declare a structure to store the following information of an student:
- a. id
- b.Student name
- c. marks
- Write a ‘C’ program to store the data of ‘n’ students, where n is given by the user.
- Write a function to display the student information getting the maximum marks.

Arrays of structures

```
void main()
{
    int i,,n ;
    printf("Enter the number of students : ") ;
    scanf("%d", &n) ;
    for(i = 0 ; i < n ; i++)
    {
        printf("\nEnter the college id : ") ;
        scanf("%d", &c[i].id ) ;
        printf("\nEnter the name : ") ;
        fflush(stdin);
        gets(c[i].name);
        printf("\nEnter the marks : ") ;
        scanf("%f", &c[i].marks) ;
    }
```

Arrays of structures

```
printf("\nId  Name \t Marks \t \n") ;
    for(i = 0 ; i < n ; i++)
    {
        printf("%d \t %s \t %f  \n", c[i].id,
        c[i].name, c[i].marks) ;
    }
```

Arrays of structures

```
void min(struct college e[10], int n )
{
    int i;
    struct college min;
    min = e[0];
    for(i = 0 ; i < n ; i++)
    {
        if(e[i].marks<min.marks)
            min = e[i];
    }
    printf("Minimum \n");
     printf("%d \t %s \t %f  \n", min.id,
        min.name, min.marks) ;
}
```

Arrays of structures

```
void main()
{
    int i,j,n ;
    struct college temp;
    printf("Enter the number of students : ") ;
    scanf("%d", &n) ;
    for(i = 0 ; i < n ; i++)
    {
        printf("\nEnter the college id : ") ;
        scanf("%d", &c[i].id ) ;
        printf("\nEnter the name : ") ;
        fflush(stdin);
        gets(c[i].name);
        printf("\nEnter the marks : ") ;
        scanf("%f", &c[i].marks) ;
    }
    printf("\nId  Name \t Marks \t \n") ;
    for(i = 0 ; i < n ; i++)
    {
        printf("%d \t %s \t %f  \n", c[i].id,
        c[i].name, c[i].marks) ;
    }
    min(c,n);
   /* sort(e,n);*/
}
```

- Write a function to display the roll no wise students records.

```
void roll_display(struct student s[],int n)
{
int i,j;
struct student temp;
for(i=0;i<=n-2;i++)
{
    for(j=i+1;j<=n-1;j++)
    {
        if(s[i].roll>=s[j].roll)
          {
                temp=s[i];
                  s[i]=s[j];
                 s[j]=temp;
          }
    }
}
for(i=0;i<n;i++)
{
           printf("\n%d\t %s\t %d\n",s[i].roll,s[i].name,s[i].marks);
}
}
```

```
Enter the no of students:3
Enter name abc
Enter roll no 3
Enter marks12
Enter name pqr
Enter roll no2
Enter marks10
Enter name xyz
Enter roll no1
Enter marks9

Max marks:12
Student records:abc 3 12

1       xyz     9

2       pqr     10

3       abc      12
```

Structures

- C program to read item details used in party and calculate all expenses, divide expenses in all friends equally.
- This program will read item name, price, quantity and calculate amount (price*quantity),
- using this program .maximum 50 items can be read and calculate the total paid amount by every person

```
//structure definition
typedef struct item_details{
        char itemName[30]; //item name
        int quantity;   //item quantity
        float price;    //per item price
        float totalAmount; //total amount = quantity*price
}item;
```

```
int main(){
        item thing[50]; //structure variable
        int i,n,n1;
        int count=0;
        float expenses=0.0;

        printf("Enter number of items");
        scanf("%d",&n);
        for(i=0;i<n;i++)
   {
                printf("Enter item details %d \n",i+1);

                printf("Item?  ");
                fflush(stdin);
                gets(thing[i].itemName);

```

```
• int main(){
•                 printf("Price? ");
•                 scanf("%f",&thing[i].price);

•                 printf("Quantity?  ");
•                 scanf("%d",&thing[i].quantity);

•
         thing[i].totalAmount=thing[i].quantity*thing[i].price;
•                 expenses += thing[i].totalAmount;
•         }
```

```
• //print all items
•         printf("All details are:\n");
•         for(i=0; i<n; i++)
•         {
•                 printf("%s\t %.2f \t %3d \n %.2f\n",thing[i].itemName,
  thing[i].price, thing[i].quantity, thing[i].totalAmount);
•         }
•         printf("#### Total expense: %.2f\n",expenses);

•         printf("Want to divide in friends");
•         printf("How many friends? ");
•                 scanf("%d",&n1);
•                 printf("Each friend will have to pay:
  %.2f\n",(expenses/(float)n1));
•         }
```

Nested Structures

```
/*C program to demonstrate example of nested structure*/
#include <stdio.h>

struct student{
        char name[30];
        int rollNo;

        struct dateOfBirth{
                int dd;
                int mm;
                int yy;
        }DOB;  /*created structure varoable DOB*/
};
```

Nested Structures

```
int main()
{
        struct student std;

        printf("Enter name: ");
        gets(std.name);
        printf("Enter roll number: ");
        scanf("%d",&std.rollNo);
        printf("Enter Date of Birth [DD MM YY] format: ");
        scanf("%d%d%d",&std.DOB.dd,&std.DOB.mm,&std.DOB.yy);
        printf("\nName : %s \nRollNo : %d \nDate of birth : 
%02d/%02d/%02d\n",std.name,std.rollNo,std.DOB.dd,std.DOB.mm,st
d.DOB.yy);

        return 0;
}
```

Difference between structure and union

- Unions are quite similar to the structures in C.
- Union is also a derived type as structure.
- Union can be defined in same manner as structures just the keyword used in defining union in **union** where keyword used in defining structure was **struct.**

```
union car
{
char name[50];
 int price;
 };
```

Difference between structure and union

```
union car
 {
   char name[50];
    int price;
}  c1, c2;
```

- The member of unions can be accessed in similar manner as that structure.
- Suppose, we you want to access price for union variable *c1* in above example, it can be accessed as `c1.price`

Difference between structure and union

- Though unions are similar to structure in so many ways, the difference between them is crucial to understand.

- #include <stdio.h>
- union job { //defining a union
- char name[32];
- float salary;
- int worker_no;
- }u;

Difference between structure and union

- struct job1 {
- char name[32];
- float salary;
- int worker_no;
- }s;

Difference between structure and union

- int main()
- {
- printf("size of union = %d",sizeof(u));
- printf("\nsize of structure = %d", sizeof(s));
- return 0;
- }

- `size of union = 32`
- `size of structure = 40`
- There is difference in memory allocation between union

  and structure as suggested in above example.

Difference between structure and union

- The amount of memory required to store a structure variables is the sum of memory size of all members.

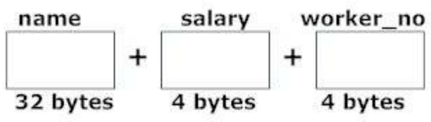

Fig: Memory allocation in case of structure

Difference between structure and union

- But, the memory required to store a union variable is the memory required for largest element of an union.

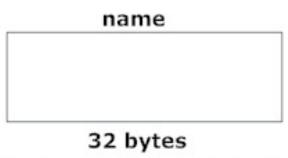

Fig: Memory allocation in case of union

Difference between structure and union

- **What difference does it make between structure and union?**
- As you know, all members of structure can be accessed at any time. But, only one member of union can be accessed at a time in case of union and other members will contain garbage value.

Difference between structure and union

```
#include <stdio.h>
union job {
 char name[32];
float salary;
 int worker_no;
}u;
```

```
• int main()
• {
•  printf("Enter name:\n");
• scanf("%s",&u.name);
• printf("Enter salary: \n");
  scanf("%f",&u.salary);
  printf("Displaying\nName :%s\n",u.name);
  printf("Salary: %.1f",u.salary);
•  return 0; }
```

```
• Enter name Hillary Enter salary 1234.23
  Displaying Name: f%Bary Salary: 1234.2
```

Difference between structure and union

- **Why this output?**
- Initially, *`Hillary`* will be stored in `u.name` and other members of union will contain garbage value.
- But when user enters value of salary, 1234.23 will be stored in `u.salary` and other members will contain garbage value.
- Thus in output, salary is printed accurately but, name displays some random string

Difference between structure and union

- You use a union when your "thing" can be one of many different things but *only one at a time.*
- You use a structure when your "thing" should be a group of other things.

- Extra Slides on Structures

Structure with pointer

- *C program to demonstrate example of structure pointer (structure with pointer)*/
- #include <stdio.h>

```
struct item
{
        char itemName[30];
        int qty;
        float price;
        float amount;
};
```

Structure with pointer

- int main()
- {
- 	struct item itm; /*declare variable of structure item*/
- 	struct item *pItem;		/*declare pointer of structure item*/
- 	pItem = &itm;			/*pointer assignment - assigning address of itm to pItem*/
- 	/*read values using pointer*/
- 	printf("Enter product name: ");
- 	gets(pItem->itemName);
- 	printf("Enter price:");
- 	scanf("%f",&pItem->price);
- 	printf("Enter quantity: ");
- 	scanf("%d",&pItem->qty);

Structure with pointer

- /*calculate total amount of all quantity*/
- 	pItem->amount =(float)pItem->qty * pItem->price;
- 	/*print item details*/
- 	printf("\nName: %s",pItem->itemName);
- 	printf("\nPrice: %f",pItem->price);
- 	printf("\nQuantity: %d",pItem->qty);
- 	printf("\nTotal Amount: %f",pItem->amount);
- 	return 0;
- }

More practice

- Create a structure to specify data of customers in a bank. The data to be stored is: Account number, Name, Balance in account. Assume maximum of 20 customers in the bank. Read all customers details
- 1. Print the Account number and name and balance of each customer.
- 2. Withdraw money
- 3. Deposit money
- 4. Search Customer

```
#include <stdio.h>

struct customer
{
    int account_no;
    char name[80];
    int balance;
};

void accept(struct customer[], int);
void display(struct customer[], int);
int search(struct customer[], int, int);
void deposit(struct customer[], int, int, int);
void withdraw(struct customer[], int, int, int);
```

```c
int main()
{
    struct customer data[20];
    int n, choice, account_no, amount, index;

    printf("Banking System\n\n");
    printf("Number of customer records you want to enter? : ");
    scanf("%d", &n);
    accept(data, n);

```

```c
do
  {

      printf("\nBanking System Menu :\n");
      printf("Press 1 to display all records.\n");
      printf("Press 2 to search a record.\n");
      printf("Press 3 to deposit amount.\n");
      printf("Press 4 to withdraw amount.\n");
      printf("Press 0 to exit\n");
      printf("\nEnter choice(0-4) : ");
      scanf("%d", &choice);
      switch (choice)
      {
          case 1:
              display(data, n);
              break;
          case 2:
              printf("Enter account number to search : ");
              scanf("%d", &account_no);
              index = search(data, n, account_no);
              if (index ==  - 1)
              {
                  printf("Record not found : ");
              }
              else
              {
                  printf("A/c Number: %d\nName: %s\nBalance: %d\n",
                      data[index].account_no, data[index].name,
                      data[index].balance);
              }
              break;
• case 3:
•               printf("Enter account number : ");
•               scanf("%d", &account_no);
•               printf("Enter amount to deposit : ");
•               scanf("%d", &amount);
•               deposit(data, n, account_no, amount);
•               break;
•           case 4:
•               printf("Enter account number : ");
•               scanf("%d", &account_no);
•               printf("Enter amount to withdraw : ");
•               scanf("%d", &amount);
•               withdraw(data, n, account_no, amount);
•       }
•    }
•    while (choice != 0);

•    return 0;
• }
```

```
• void accept(struct customer list[80], int s)
• {
•     int i;
•     for (i = 0; i < s; i++)
•     {
•         printf("\nEnter data for Record #%d", i + 1);

•         printf("\nEnter account_no : ");
•         scanf("%d", &list[i].account_no);
•         fflush(stdin);
•         printf("Enter name : ");
•         gets(list[i].name);
•         list[i].balance = 0;
•     }
• }
```

```
void display(struct customer list[80], int s)
{
    int i;

    printf("\n\nA/c No\tName\tBalance\n");
    for (i = 0; i < s; i++)
    {
        printf("%d\t%s\t%d\n", list[i].account_no, list[i].name,
            list[i].balance);
    }
}
```

```
int search(struct customer list[80], int s, int number)
{
    int i;

    for (i = 0; i < s; i++)
    {
        if (list[i].account_no == number)
        {
            return i;
        }
    }
    return  - 1;
}
```

```
void deposit(struct customer list[], int s, int number, int amt)
{
    int i = search(list, s, number);
    if (i ==  - 1)
    {
        printf("Record not found");
    }
    else
    {
        list[i].balance += amt;
    }
}
```

```
void withdraw(struct customer list[], int s, int number, int amt)
{
    int i = search(list, s, number);
    if (i ==  - 1)
    {
        printf("Record not found\n");
    }
    else if (list[i].balance < amt)
    {
        printf("Insufficient balance\n");
    }
    else
    {
        list[i].balance -= amt;
    }
}
```