Structures

Structures (1)

- Structures are C’s way of grouping collections of data into a single manageable unit.
  - in which the individual elements can differ in type
  - a single structure might contain integer elements, floating-point elements ,character elements,pointers, arrays and other structures
  - The individual structure elements are referred to as ***members.***
  - Defining a structure type:

```
struct coord {
        int x ;
        int y ;
};
```

  - This defines a new type struct coord. No variable is actually declared or generated.

Structures (2)

- Define struct variables:

```
struct coord {
       int x,y ;
  } first, second;
```

- Another Approach:

```
struct coord {
      int x,y ;
 };
...............
struct coord first, second;  /* declare variables */
struct coord third;
```

USER-DEFINED DATA TYPES (typedef)

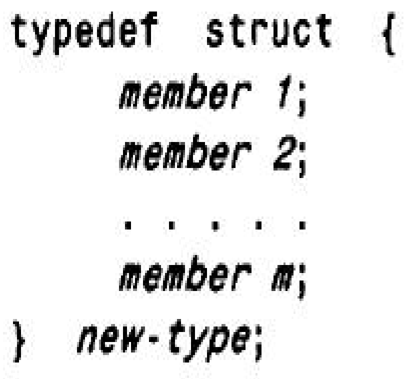
- The **typedef** feature allows users to define new data-types that are equivalent to existing data types.
- Once a user-defined data type has been established, then new variables, arrays, structures, etc. can be declared in terms of this new data type.
- In general terms, a new data type is defined **as**

  typedef *type new- type;*

- where ***type*** refers to an existing data type (either a standard data type, or previous user-defined data type), and ***new- type*** refers to the new user-defined data type

Examples

- **typedef int age;// defines a userdefined datatype age of type int**
- **age x,y;**
- **Structure type**



```
typedef  struct  {
      member 1;
      member 2;
      . . . . .
      member m;
}  new-type;
```

where ***new- type*** is the user-defined structure type. Structure variables can then be defined in terms of the new data type.

Structures (3)

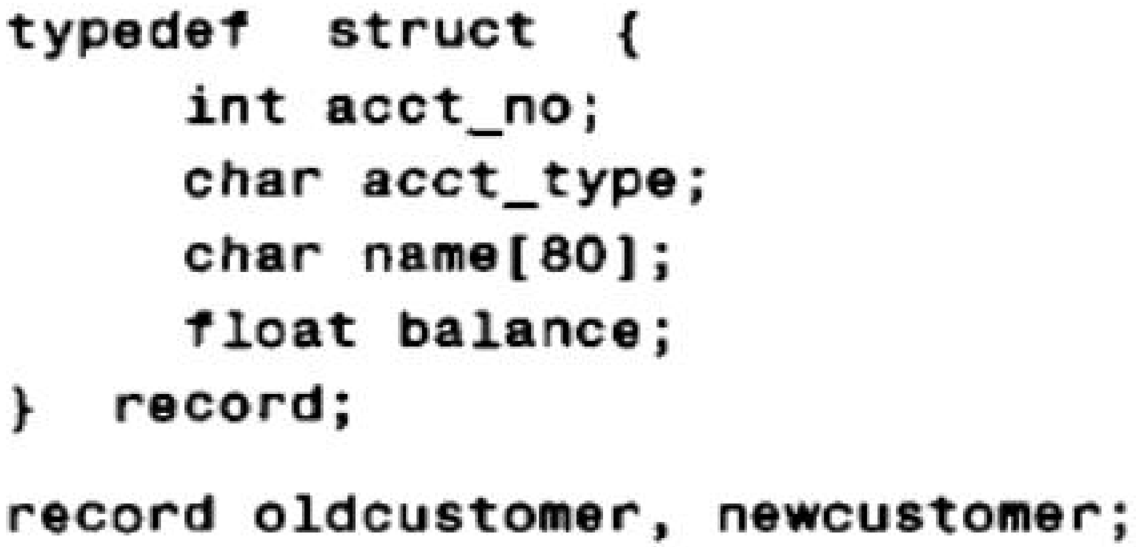
- You can even use a typedef if your don't like having to use the word “struct”

```
typedef struct coord coordinate;
coordinate first, second;
```

```
                    typedef struct {int a, b; char *p;} S;
/* omit both tag and variables */
```

  - This creates a simple type name S

Example



```
typedef  struct  {
     int acct_no;
     char acct_type;
     char name[80];
     float balance;
}  record;

record oldcustomer, newcustomer;
```

Structures (5)

- You can assign structures as a unit with =

```
    first = second;
```

  instead of writing:

```
     first.x = second.x ;
     first.y = second.y ;
```

- Although the saving here is not great
  - It will reduce the likelihood of errors and
  - Is more convenient with large structures

Structures (4)

- Access structure variables by the dot (.) operator
- Generic form:

```
structure_var.member_name
```

- For example:

```
first.x = 50 ;
second.y = 100;
```

- struct_var.member_name can be used anywhere a variable can be used:
  - printf ("%d , %d", second.x , second.y);
  - scanf("%d, %d", &first.x, &first.y);

Structures Containing Structures

- Any "type" of thing can be a member of a structure.
- We can use the coord struct to define a rectangle

```
struct rectangle {
   struct coord topleft;
   struct coord bottomrt;
} ;
```

- This describes a rectangle by using the two points necessary:

```
struct rectangle mybox ;
```

- Initializing the points:

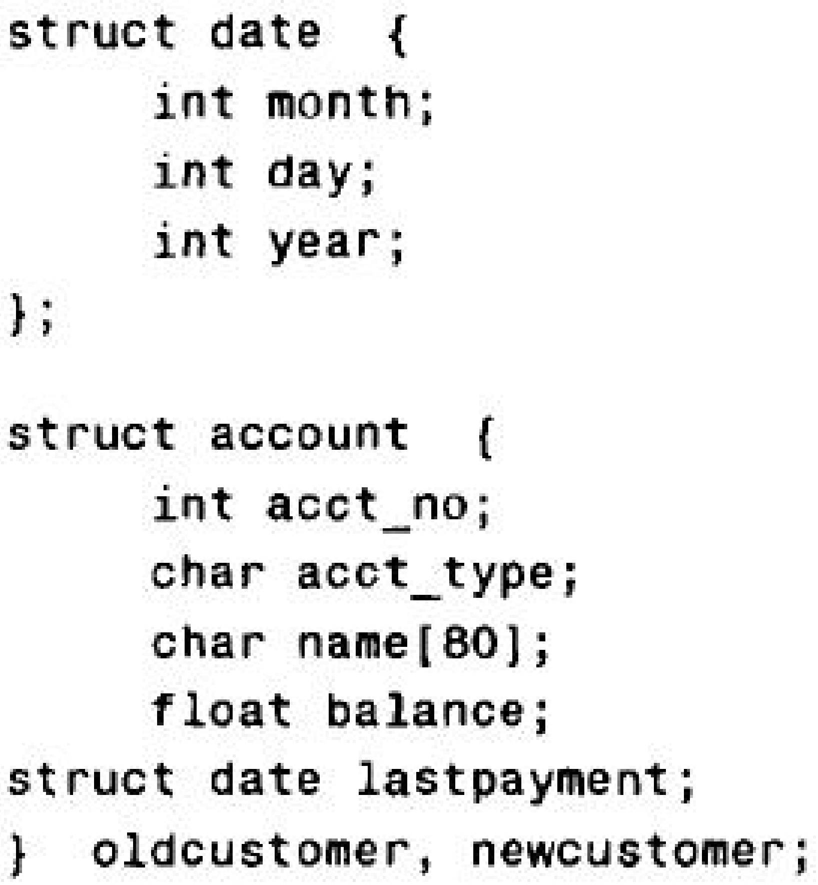
```
mybox.topleft.x = 0 ;
mybox.topleft.y = 10 ;
mybox.bottomrt.x = 100 ;
mybox.bottomrt.y = 200 ;
```

An Example

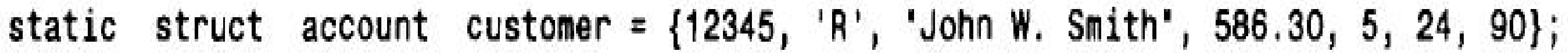
```
#include <stdio.h>
struct coord {
    int x;
    int y;
};
struct rectangle {
    struct coord topleft;
    struct coord bottomrt;
};
```

```
int main () {
    int length, width;
    long area;
    struct rectangle mybox;
    mybox.topleft.x = 0;
    mybox.topleft.y = 0;
    mybox.bottomrt.x = 100;
    mybox.bottomrt.y = 50;
    width = mybox.bottomrt.x –
                mybox.topleft.x;
    length = mybox.bottomrt.y –
                mybox.topleft.y;
    area   = width * length;
    printf ("The area is %ld units.\n", 
    area);
}
```

More examples



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

struct account  {
     int acct_no;
     char acct_type;
     char name[80];
     float balance;
struct date lastpayment;
}  oldcustomer, newcustomer;
```



```
static  struct  account  customer = {12345, 'R', 'John W. Smith', 586.30, 5, 24, 90};
```

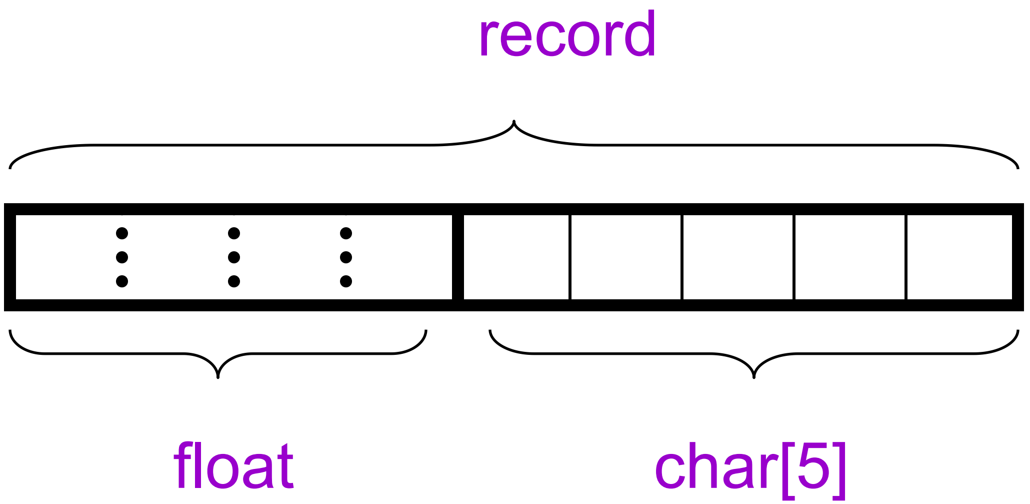
customer.lastpayment.month

Structures Containing Arrays

- Arrays within structures are the same as any other member element.
- For example:

```
struct record {
        float  x;
        char  y [5] ;
  } ;
```

- Logical organization:



An Example

```
#include <stdio.h>
struct data {
   int mark;
   char  fname[30];
   char lname[30];
}
int main () {
    struct data student;
    printf ("Enter the student r's first and last names, \n");
           scanf  ("%s %s", student.fname, student.lname);
    printf ("\nEnter the mark of subject: ");
    scanf  ("%d", & student.mark);
printf ("\nstudent %s %s scored $%d marks for  the subject \n",
            student.fname,student.lname,student.mark);
}
```

An Example

```
#include <stdio.h>
struct data {
    int mark[5];
    char  fname[30];
    char lname[30];
}
int main () {
    struct data student;
int i;
    printf ("Enter the student r's first and last names, \n");
            scanf  ("%s %s", student.fname, student.lname);
for( i=0;i<5;i++){
    printf ("\nEnter the mark of subject: %d“, i+1);
    scanf  ("%d", &student.mark[i]);
printf ("\nstudent %s %s scored $%d marks for  the subject %d \n",
             student.fname,student.lname,student.mark[i]);}
}
```

Arrays of Structures

- The converse of a structure with arrays:
- Example:

```
struct  data  {
   char  fname [10] ;
   char  lname [12] ;
   int  marks [5] ;
} ;
struct  stud_rec [60];
```

- This creates student records of 60 identical entry(s).
- Assignments:

```
stud_rec [1] = stud_rec [6];
strcpy (stud_rec [1].marks, stud_rec [6].marks);
stud_rec [6].marks[1] = stud_rec [3].marks[4] ;
```

Phone list

```
#include <stdio.h>
struct entry {
    char fname [20];
    char lname [20];
    char phone [10];
} ;
```

```
int main() {
   struct entry list[60];
   int i;
   for (i=0; i < 60; i++) {
       printf ("\nEnter first name: ");
       scanf  ("%s", list[i].fname);
       printf ("Enter last name: ");
       scanf  ("%s", list[i].lname);
       printf ("Enter phone in 123-4567 format: ");
       scanf  ("%s", list[i].phone);
   }
   printf ("\n\n");
   for (i=0; i < 60; i++) {
       printf ("Name: %s %s", list[i].fname, list[i].lname);
       printf ("\t\tPhone: %s\n", list[i].phone);
   }
}
```

Initializing Structures

- struct point
- {
- int x = 0; // COMPILER ERROR: cannot initialize members here
- int y = 0; // COMPILER ERROR: cannot initialize members here
- };

struct point p1 = { 0,0 } ;

Initializing Structures

- Simple example:

```
struct sale {
  char  customer [20] ;
  char  item [20] ;
  int amount ;
};

struct sale mysale = { "Acme Industries",
                       "Zorgle blaster",
                       1000 } ;
```

Initializing Structures

- Structures within structures:

```
struct  customer {
   char firm [20] ;
   char contact [25] ;
};
struct sale {
   struct customer buyer ;
   char item [20] ;
   int amount ;
} mysale =
{ { "Acme Industries", "George Adams"} ,
    "Zorgle Blaster",  1000
} ;
```

Initializing Structures

- Arrays of structures

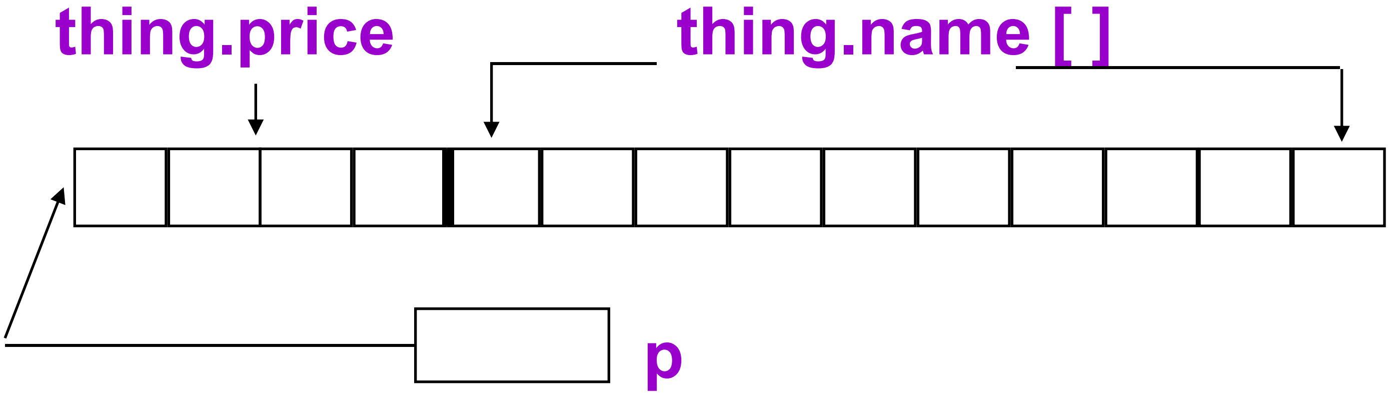
```
struct customer {
   char firm [20] ;
   char contact [25] ;
} ;
struct sale {
   struct customer buyer ;
   char item [20] ;
   int amount ;
} ;
```

```
struct sale y1990 [100] = {
   {  { "Acme Industries",
   "George Adams"} ,
   "Left-handed Idiots" ,
   1000
   },
   {  { "Wilson & Co.",
        "Ed Wilson"} ,
   "Thingamabob" , 290
   }
} ;
```

Pointers to Structures

```
struct part {
   float price ;
   char name [10] ;
} ;
struct part *p , thing;
p = &thing;
/* The following three statements are equivalent */
thing.price = 50;
(*p).price = 50;   /* () around *p is needed */
p -> price = 50;
```

Pointers to Structures



- p is set to point to the first byte of the struct variable

Pointers to Structures

```
struct part * p, *q;
p = (struct part *) malloc( sizeof(struct part) );
q = (struct part *) malloc( sizeof(struct part) );
p -> price = 199.99 ;
strcpy( p -> name, "hard disk" );
(*q) = (*p);
q = p;
free(p);
free(q); /* This statement causes a problem !!!
                Why? */
```

Pointers to Structures

- You can allocate a structure array as well:

```
{
    struct part *ptr;
    ptr = (struct part *) malloc(10 * sizeof(struct part) );
    for( i=0; i< 10; i++)
    {
        ptr[ i ].price = 10.0 * i;
        sprintf( ptr[ i ].name, "part %d", i );
    }
    ……
    free(ptr);
}
```

- String print function it is stead of printing on console store it on char buffer which are specified in sprint. // Example program to demonstrate sprintf()

```
#include<stdio.h>
int main()
{
    char buffer[50];
    int a = 10, b = 20, c;
    c = a + b;
    sprintf(buffer, "Sum of %d and %d is %d", a, b, c);

    // The string "sum of 10 and 20 is 30" is stored
    // into buffer instead of printing on stdout
    printf("%s", buffer);

    return 0;
}
```

Pointers to Structures

- You can use pointer arithmetic to access the elements of the array:

```
{
    struct part *ptr,  *p;
    ptr = (struct part *) malloc(10 * sizeof(struct part) );
    for( i=0, p=ptr; i< 10; i++, p++)
    {
        p -> price = 10.0 * i;
        sprintf( p -> name, "part %d", i );
    }
    ……
    free(ptr);
}
```

Self referential structures

- A **self referential structure** is essentially a **structure** definition which includes at least one member that is a pointer to the **structure** of its own kind.

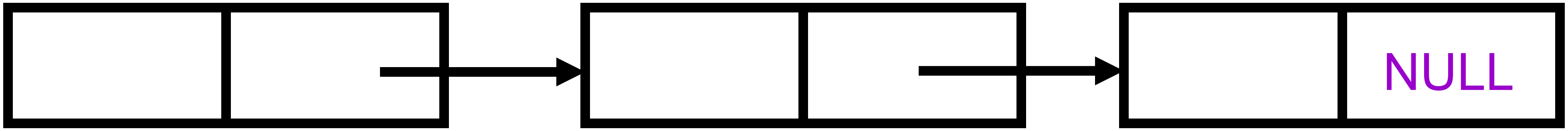
-
  ```
  struct struct_name
  {
  datatype datatypename;
  struct_name * pointer_name;
  };
  ```

- A self referential structure is used to create data structures like linked lists, stacks, etc

Self referential structures(Pointer as Structure Member)

```
struct node{
   int data;
   struct node *next;
};
struct node a,b,c;
a.next = &b;
b.next = &c;
c.next = NULL;
```

```
a.data = 1;
a.next->data = 2;
/* b.data =2 */
a.next->next->data = 3;
/* c.data = 3 */
c.next = (struct node *)
   malloc(sizeof(struct
   node));
……
```



Assignment Operator vs. memcpy

- This assign a struct to another

```
{
    struct part a,b;
    b.price = 39.99;
    b.name = "floppy";
    a = b;
}
```

- Equivalently, you can use memcpy

```
#include <string.h>
……
{
    struct part a,b;
    b.price = 39.99;
    b.name = "floppy";
    memcpy(&a,&b,sizeof(part));
}
```

Array Member vs. Pointer Member

```
struct book {
    float price;
    char name[50];
};
```

```
int main()
{
    struct book a,b;
    b.price = 19.99;
    strcpy(b.name, "C handbook");
    a = b;
    strcpy(b.name, "Unix handbook");
    puts(a.name);
    puts(b.name);
}
```

Array Member vs. Pointer Member

```
struct book {
    float price;
    char *name;
};
```

```
int main()
{
    struct book a,b;
    b.price = 19.99;
    b.name = (char *) malloc(50);
    strcpy(b.name, "C handbook");
    a = b;
    strcpy(b.name, "Unix handbook");
    puts(a.name);
    puts(b.name);
    free(b.name);
}
```

A function called strdup() will do the malloc() and strcpy() in one step for you!

Passing Structures to Functions (1)

- Several different ways to pass structure–type information to or from a function.
- Structure member can be transferred individually , or entire structure can be transferred.
- The individual structures members can be passed to a function as arguments in the function call; and a single structure member can be returned via the return statement.
- To do so, each structure member is treated the same way as an ordinary, single- valued variable.

Passing BY VALUE(1)

- This means that the structure is copied if it is passed as a parameter.
  - This can be inefficient if the structure is big.
    - In this case it may be more efficient to pass a pointer to the struct.
- A struct can also be returned from a function.

Pass by reference(1)

- A complete structure can be transferred to a function by passing a structure type pointer as an argument.
- A structure passed in this manner will be passed by reference rather than by value.
- So, if any of the structure members are altered within the function, the alterations will be recognized outside the function.

Pass by reference-Passing structure pointers to functions

```
# include <stdio.h>
typedef struct{
char *name;
int roll_no;
float marks ;
} record ;
main ( )
{
void adj(record *ptr);
static record stduent={“Anu”,2,99.9};
printf(“%s%d%f\n”, student.name,
student.roll_no,student.marks);
adj(&student);
printf(“%s%d%f\n”, student.name,
student.roll_no,student.marks);
}
```

```
void adj(record*ptr)
{
Ptr-> name=”Binu”;
ptr -> roll_no=3;
ptr -> marks=98.0;
return;
}
```

Pass by reference-Passing entire structure to function

```
# include <stdio.h>
typedef struct{
char *name;
int roll_no;
float marks ;
} record ;
main ( )
{
void adj(record stud);
static record student={“Anu”,2,99.9};
printf(“%s%d%f\n”, student.name,
student.roll_no, student.marks);
adj(student);
printf(“%s%d%f\n”, student.name,
student.roll_no, student.marks);
}
```

```
void adj(record stud)
{
stud.name=”Binu”;
stud. roll_no=3;
stud. marks=98.0;
return;
}
```

A struct can also be returned from a function

```
struct pairInt {
    int min, max;
};
struct pairInt min_max(int x,int y)
{
    struct pairInt pair;
    pair.min = (x > y) ? y : x;
    pair.max = (x > y) ? x : y;
    return pairInt;
}
int main(){
    struct pairInt result;
    result = min_max( 3, 5 );
    printf("%d<=%d", result.min, result.max);
}
```

union

- A structure is a user-defined data type available in C that allows to combining data items of different kinds. Structures are used to represent a record.
- A union is a special data type available in C that allows storing different data types in the same memory location.
- You can define a union with many members, but only one member can contain a value at any given time.
- Unions provide an efficient way of using the same memory location for multiple purposes.

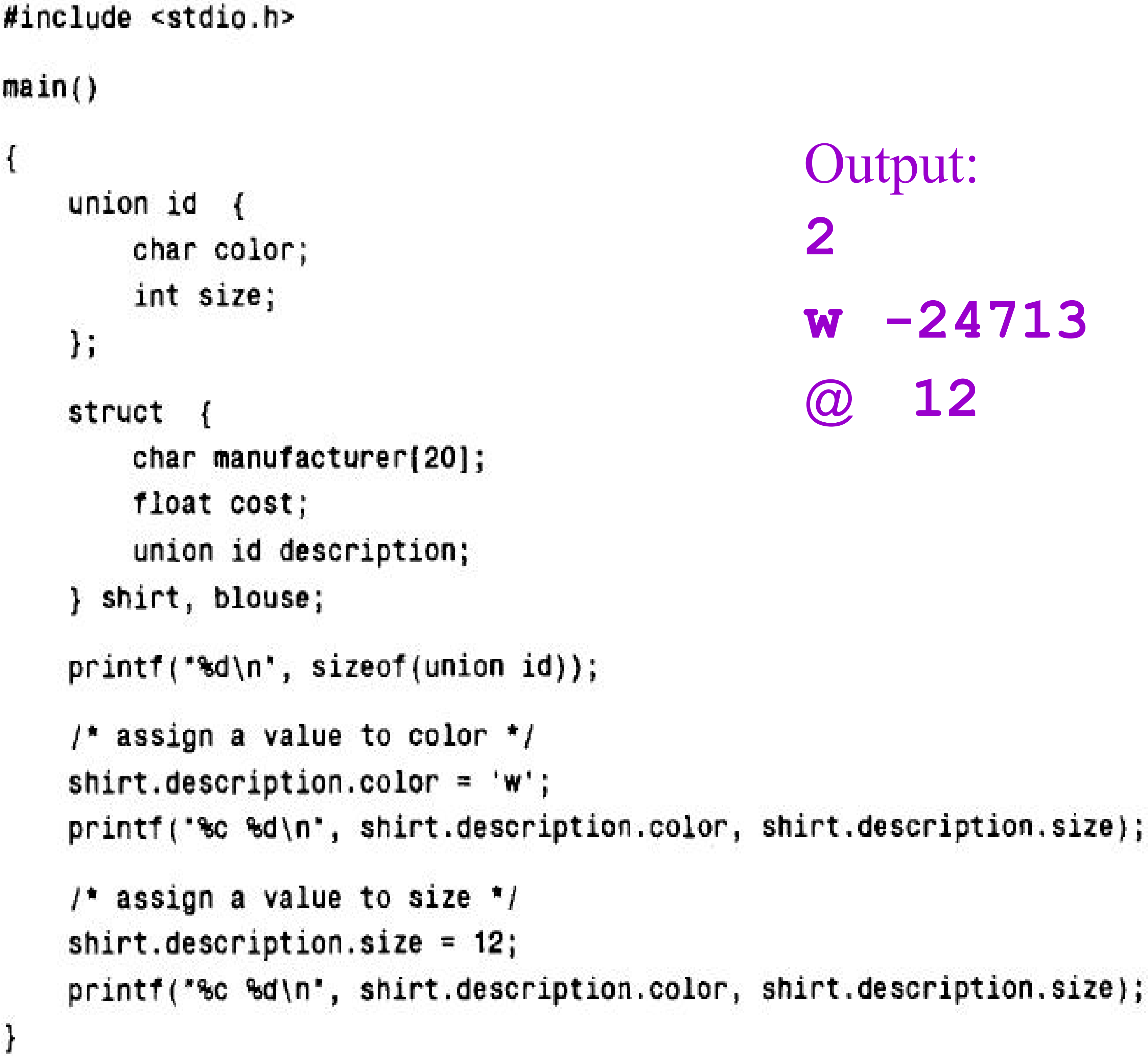
Defining a Union

- union union_name

{ member definition;

 member definition; ...

 member definition; };

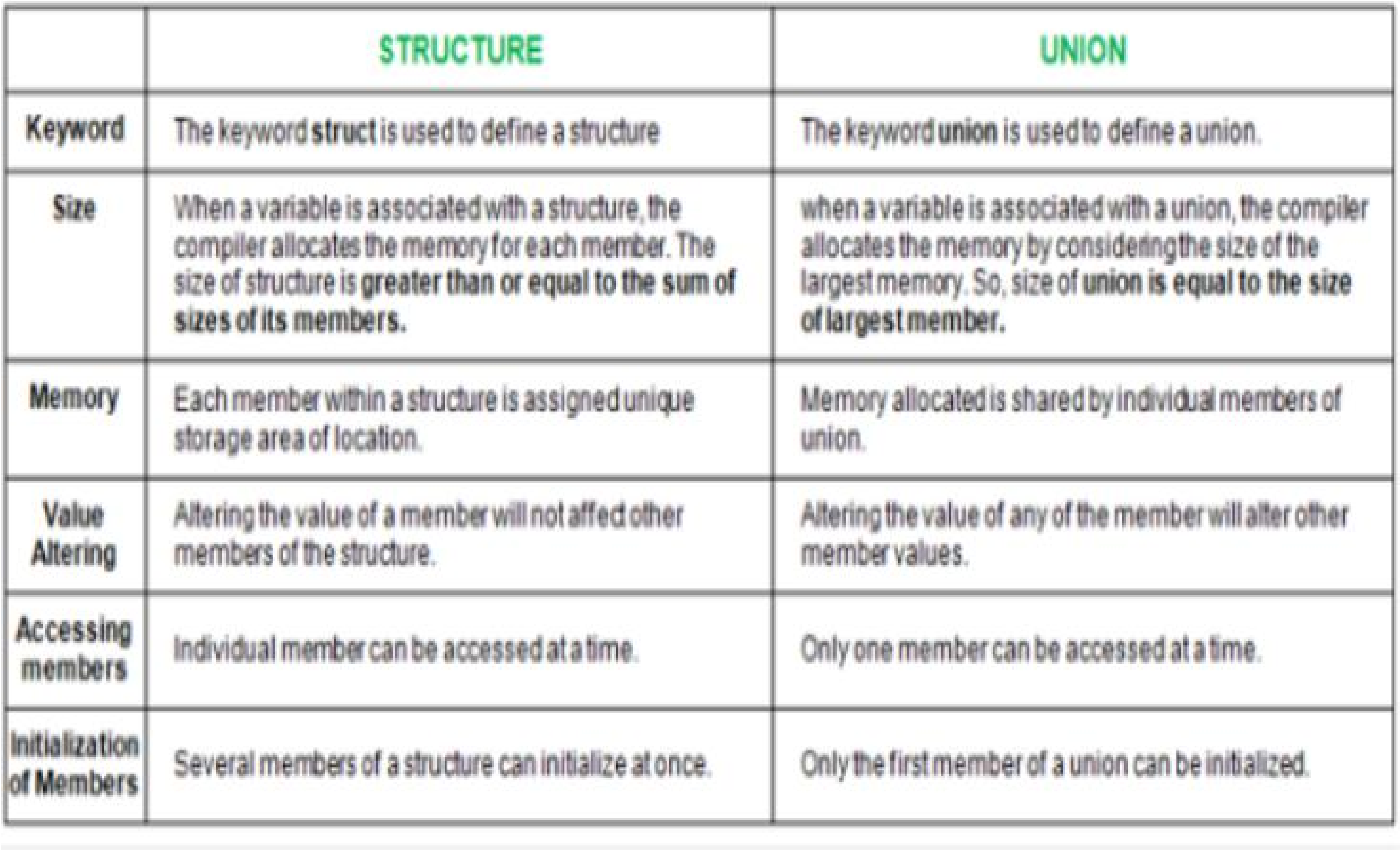


```c
#include <stdio.h>

main()

{
    union id  {
        char color;
        int size;
    };

    struct  {
        char manufacturer[20];
        float cost;
        union id description;
    } shirt, blouse;

    printf("%d\n", sizeof(union id));

    /* assign a value to color */
    shirt.description.color = 'w';
    printf("%c %d\n", shirt.description.color, shirt.description.size);

    /* assign a value to size */
    shirt.description.size = 12;
    printf("%c %d\n", shirt.description.color, shirt.description.size);
}
```

Output:

```
2
w -24713
@  12
```

*Differences between structure and union



	STRUCTURE	UNION
Keyword	The keyword **struct** is used to define a structure	The keyword **union** is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is **greater than or equal to the sum of sizes of its members.**	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of **union is equal to the size of largest member.**
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only the first member of a union can be initialized.

- Program to read 10 student records (rollno ,marks obtained in two subjects,) and print total and average marks each student

```
#include<stdio.h>
struct stud
{
   int rollno, m1, m2, tot ;
    char name[10] ;
    float avg ;
} s[10] ;
void main()
{
    int i, n ;
    clrscr() ;
    printf("Enter the number of
students : ") ;
scanf("%d", &n) ; //n-=10
for(i = 0 ; i < n ; i++)
    {
printf("\nEnter the roll number : ") ;
 scanf("%d", &s[i].rollno) ;
printf("\nEnter the name : ") ;
scanf("%s", s[i].name) ;
printf("\nEnter the marks in 2 subjects : ") ;
scanf("%d %d", &s[i].m1, &s[i].m2) ;
 s[i].tot = s[i].m1 + s[i].m2 ;
        s[i].avg = s[i].tot / 2.0 ;
    }
    printf("\nRoll No. Name \t\tSub1\t Sub2\t
Total\t Average\n\n") ;
    for(i = 0 ; i < n ; i++)
    {
   printf("%d \t %s \t\t %d \t %d \t %d \t %.2f \n",
s[i].rollno, s[i].name, s[i].m1, s[i].m2, s[i].tot,
s[i].avg);
    }
}
```

Printing Marksheets of students

```
struct mark_sheet {
    char name[20];
     long int rollno;
     int marks[10];
      int total;
       float average;
       char rem[10];
        char cl[20];
        }students[60];
int main(){
       int a,b,n,flag=1;
       char ch;
     printf("How many students : \n");
     scanf("%d",&n);
     for(a=1;a<=n;++a){
printf("\n\nEnter the details of %d students : 
", n-a+1);
printf("\n\nEnter student  %d  Name : ", a);
      scanf("%s", students[a].name);
```

```
printf("\n\nEnter student %d Roll Number : 
", a);
    scanf("%ld",& students[a].rollno);
     students[a].total=0;
       for(b=1;b<=5;++b)
{  printf("\n\nEnter the mark of subject-%d : 
 ", b);
   scanf("%d",& students[a].marks[b]);
  students[a].total += students[a].marks[b];
     if(students[a].marks[b]<40)
            flag=0;
            }
students[a].average = 
(float)(students[a].total)/5.0;

if((students[a].average>=75)&&(flag==1))
      strcpy(students[a].cl,"Distinction");
            else

if((students[a].average>=60)&&(flag==1))
    strcpy(students[a].cl,"First Class");
```

```
if((students[a].average>=50)&&(flag==1))
    strcpy(students[a].cl,"Second Class");
        else
if((students[a].average>=40)&&(flag==1))
        strcpy(students[a].cl,"Third Class");
            if(flag==1)
      strcpy(students[a].rem,"Pass");
            else
            strcpy(students[a].rem,"Fail");
            flag=1;
            }
for(a=1;a<=n;++a){
            clrscr();
       printf("\n\n\t\t\t\tMark Sheet\n");
     printf("\nName of Student : %s",
students[a].name);
            printf("\t\t\t\t Roll No : %ld",
students[a].rollno);
```

```
printf("\n-----------------------------------");
            for(b=1;b<=5;b++){
 printf("\n\n\t Subject %d \t\t :\t %d", b,
students[a].marks[b]);
             }
 printf("\n\n---------------------\n");
 printf("\n\n Totl Marks : %d",
students[a].total);
 printf("\t\t\t\t Average Marks : %5.2f",
students[a].average);
 printf("\n\n Class : %s",
students[a].cl);
             printf("\t\t\t\t\t Status : %s",
students[a].rem);
 printf("\n\n\n\t\t\t\t Press Y for continue
. . . ");
             ch = getchar();
             if((ch=="y")||(ch=="Y"))
             continue;
             }
             return(0);
             }
```

Sample output

```
Mark Sheet

Name of Student :  Hari                    Roll No :  536435
----------------------------------------------------------------------

                    subject  1              :  46
                    subject  2              :  56
                    subject  3              :  76
                    subject  4              :  85
                    subject  5              :  75
 ----------------------------------------------------------------------
 Totl Marks :      338                       Average Marks :  67.6

Class :          First Class                    Status :  Pass



                                       Press Y for continue . . .
```

homework

- The annual examination is conducted for 10 students for three subjects. Write a program to read the data and determine the following:
  (a) Total marks obtained by each student.
  (b) The highest marks in each subject and the marks. of the student who secured it.
  (c) The student who obtained the highest total marks and print his /her progress sheet