STRUCTURES AND UNIONS

DoCSE

Introduction

- **It is a convenient tool for handling a group of logically related data items.**
  - **Student name, roll number, and marks**
  - **Real part and complex part of a complex number**
- **This is our first look at a non-trivial data structure.**
  - **Helps in organizing complex data in a more meaningful way.**
- **The individual structure elements are called members.**

Defining a Structure

- **The composition of a structure may be defined as:**

```
struct tag {
        member 1;
        member 2;
        :
        member m;
    };
```

  - **struct is the required keyword.**
  - **tag is the name of the structure.**
  - **member 1, member 2, … are individual member declarations.**

Contd…

- **The individual members can be ordinary variables, pointers, arrays, or other structures.**
  - **The member names within a particular structure must be distinct from one another.**
  - **A member name can be the same as the name of a variable defined outside of the structure.**
- **Once a structure has been defined, individual structure-type variables can be declared as:**

```
struct tag variable_1, variable_2, …, variable_n;
```

Example

- **A structure definition:**

```
struct student {
        char name[30];
        int roll_number;
        int total_marks;
        char dob[10];
};
```

- **Defining structure variables:**

```
struct student a1, a2, a3;
```

A new data-type

A Compact Form

- It is possible to combine the declaration of the structure with that of the structure variables:

```
struct tag {
        member 1;
        member 2;
        :
        member m;
      } variable_1, variable_2,..., variable_n;
```

- In this form, "tag" is optional.

```
struct student {
        char name[30];
        int roll_number;
        int total_marks;
        char dob[10];
      } a1, a2, a3;
```

```
struct {
        char name[30];
        int roll_number;
        int total_marks;
        char dob[10];
      } a1, a2, a3;
```

Size of a Structure

```
#include<stdio.h>

typedef struct{
    char name[80];
    char dept[30];
    int roll_no;
    } studinfo;

main()
{
studinfo a;
printf("a store %d bytes",sizeof(a));
}
```

```
struct date {
char name[80];
int month;
int day;
int year;
}

struct date birthday[ ] = {“Amy”, 12, 30, 73, “Gail”, 5, 13, 66, “Marc”, 7, 15,
    72, “Marla”, 11, 29, 70,”Megan”, 2, 4, 77,”Sharonw”,12, 29, 63, “Susan”, 4,
    12,69};
```

```
#include<stdio.h>

struct studinfo{
     char name[80];
     char dept[30];
     int roll_no;
     };

main()
{
struct studinfo si;
printf("Your name:");
scanf(“ %[^\n]",si.name);
printf("\nYour department:");
scanf("%s",si.dept);
printf("\nYour roll number:");
scanf("%d",&si.roll_no);
printf("**************************");
printf("\nYour Name:%s",si.name);
printf("\nYour Department:%s",si.dept);
printf("\nYour Roll Number:%d",si.roll_no);
}
```

```
#include<stdio.h>

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

struct studinfo{
    char name[80];
    char dept[30];
    int roll_no;
    struct date birth;
    };
```

```
main( )
{
struct studinfo si;
printf("Your name:");
scanf(“ %[^\n]",si.name);
printf("\nYour department:");
scanf("%s",si.dept);
printf("\nYour roll number:");
scanf("%d",&si.roll_no);
printf("\nDate of birth:(mm/dd/yy)");
scanf("%d%d%d",&si.birth.month,&si.birth.day,&si.birth.year);
printf("*****************************");
printf("\nYour Name:%s",si.name);
printf("\nYour Department:%s",si.dept);
printf("\nYour Roll Number:%d",si.roll_no);
printf("\nYour Birthday:%d-%d-
%d",si.birth.month,si.birth.day,si.birth.year);
}
```

```
#include<stdio.h>

struct studinfo{
    char name[80];
    char dept[30];
    int roll_no;
    };

main()
{
int n,i;
struct studinfo si[100];
printf("How many student
    information are you storing: ");
scanf("%d",&n);
for(i=0;i<n;++i){
    printf("Your name:");
    scanf(" %[^\n]",si[i].name);
    printf("Your department:");
    scanf(" %[^\n]",si[i].dept);
    printf("Your roll number:");
    scanf("%d",&si[i].roll_no);
    }
printf("\n*****************************\n");
printf("*****************************");
printf("\n");
for(i=0;i<n;i++)
            {
     printf("\nYour Name:%s",si[i].name);
     printf("\nYour Department:%s",si[i].dept);
     printf("\nYour Roll Number:%d",si[i].roll_no);
printf("\n*****************************");
            }
}
```

```
#include<stdio.h>

struct studinfo{
    char name[80];
    char dept[30];
    int roll_no;
    };

main()
{
int n,i;
struct studinfo si[100];
printf("How many student
    information are you storing: ");
scanf("%d",&n);
for(i=0;i<n;++i){
    printf("Your name:");
    scanf(" %[^\n]",si[i].name);
    printf("Your department:");
    scanf(" %[^\n]",si[i].dept);
    printf("Your roll number:");
    scanf("%d",&si[i].roll_no);
    }
printf("\n*****************************\n");
printf("*****************************");
printf("\n");
for(i=0;i<n;i++)
            {
     printf("\nYour Name:%s",si[i].name);
     printf("\nYour Department:%s",si[i].dept);
     printf("\nYour Roll Number:%d",si[i].roll_no);
printf("\n*****************************");
            }
}
```

```
#include<stdio.h>

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

struct studinfo{
    char name[80];
    char dept[30];
    int roll_no;
    struct date birth;
    };
```

```
main()
{
int n,i;
struct studinfo si[100];
printf("How many student information are you
    storing: ");
scanf("%d",&n);
for(i=0;i<n;++i){
    printf("Your name:");
    scanf(“ %[^\n]",si[i].name);
    printf("Your department:");
    fflush(stdin);
    scanf(“ %[^\n]",si[i].dept);
    printf("Your roll number:");
    scanf("%d",&si[i].roll_no);
    printf("\nDate of birth:(mm/dd/yy)");
    scanf("%d%d%d",&si[i].birth.month,&si[i].birth.d
    ay,&si[i].birth.year);
    }
```

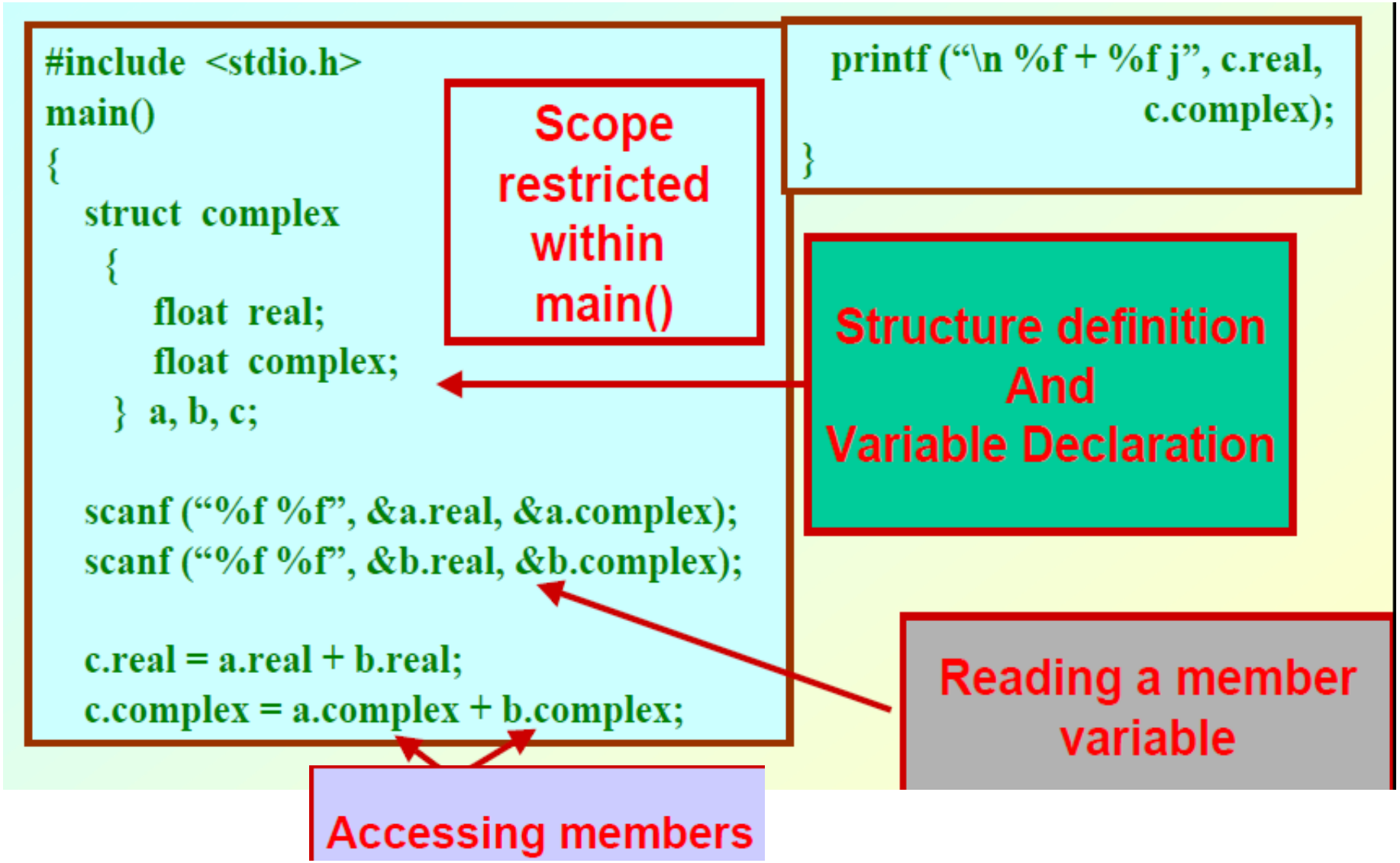
Processing a Structure

- The members of a structure are processed individually, as separate entities.
- A structure member can be accessed by writing

  variable.member

  where variable refers to the name of a structure-type variable, and member refers to the name of a member within the structure.
- Examples:
  - a1.name, a2.name, a1.roll_number, a3.dob;

Programming Example



Comparision of structure variable

- **Unlike arrays, group operations can be performed with structure variables.**
  - **A structure variable can be directly assigned to another structure variable of the same type.**

    a1 = a2;

    - **All the individual members get assigned.**
  - **Two structure variables can be compared for equality or inequality.**

    if (a1 = = a2) …….

    - **Compare all members and return 1 if they are equal; 0 otherwise.**

Arrays of Structure

- **Once a structure has been defined, we can declare an array of structures.**

```
struct student class[50];
```

  - **The individual members can be accessed as:**
    - **class[i].name**
    - **class[5].roll_number**

- **A structure member can be an array:**

```
struct student {
        char name[30];
        int roll_number;
        int marks[5];
        char dob[10];
    } a1, a2, a3;
```

- **The array element within the structure can be accessed as:**

```
a1.marks[2]
```

Defining date type: using typedef

- One may define a structure data-type with a single name.
- General syntax:

```
typedef struct {
        member-variable1;
        member-variable2;
            .
        member-variableN;
    } tag;
```

- tag is the name of the new data-type.

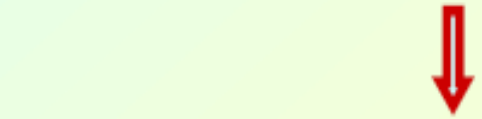
```
typedef struct{
        float real;
        float imag;
        } _COMPLEX;

_COMPLEX a,b,c;
```

Structure Initialization

- Structure variables may be initialized following similar rules of an array. The values are provided within the second braces separated by commas.
- An example:

```
COMPLEX a={1.0,2.0}, b={-3.0,4.0};
```



```
a.real=1.0; a.imag=2.0;
b.real=-3.0; b.imag=4.0;
```

Parameter Passing in a function

- **Structure variables could be passed as parameters like any other variable. Only the values will be copied during function invokation.**

```
void swap(_COMPLEX a, _COMPLEX b)
{
  _COMPLEX tmp;

  tmp=a;
  a=b;
  b=tmp;
}
```

```c
#include <stdio.h>

typedef struct{
        float real;
        float imag;
      } _COMPLEX;

  void swap(_COMPLEX a, _COMPLEX b)
   {
     _COMPLEX tmp;

     tmp=a;
     a=b;
     b=tmp;
   }
```

```c
void print(_COMPLEX a)
{
 printf("(%f , %f) \n",a.real,a.imag);
}

main()
{
 _COMPLEX x={4.0,5.0},y={10.0,15.0};

  print(x); print(y);
  swap(x,y);
  print(x); print(y);
}
```

Returning Structure

- It is also possible to return structure values from a function. The return data type of the function should be as same as the data type of the structure itself.

```
_COMPLEX add(_COMPLEX a, _COMPLEX b)
{
  _COMPLEX tmp;

  tmp.real=a.real+b.real;
  tmp.imag=a.imag+b.imag;

  return(tmp);
}
```

Direct arithmetic operations are not possible with Structure variables.

Array of Structure

```
struct stud {
        int roll;
        char dept_code[25];
        float cgpa;
    };
struct stud a, b, c;
```

- And the individual structure elements can be accessed as:

  a.roll , b.roll , c.cgpa , etc.

- We can define an array of structure records as

  struct stud class[100] ;

- The structure elements of the individual records can be accessed as:

  class[i].roll

  class[20].dept_code

  class[k++].cgpa

Example: Sorting

```
#include <stdio.h>
struct  stud
{
    int   roll;
    char  dept_code[25];
    float  cgpa;
};

main()
{
    struc  stud  class[100], t;
    int  j, k, n;

    scanf  ("%d", &n);
                /* no. of students */
for  (k=0; k<n; k++)
      scanf ("%d %s %f", &class[k].roll,
          class[k].dept_code, &class[k].cgpa);
   for  (j=0; j<n-1; j++)
     for  (k=j+1; k<n; k++)
       {
         if  (class[j].roll > class[k].roll)
           {
             t = class[j] ;
             class[j] = class[k] ;
             class[k] = t
           }
       }
  <<<< PRINT THE RECORDS >>>>
}
```

Pointer and Structure

- **You may recall that the name of an array stands for the address of its zero-th element.**
  - **Also true for the names of arrays of structure variables.**
- **Consider the declaration:**

```
struct  stud  {
                int   roll;
                char  dept_code[25];
                float  cgpa;
              }  class[100],  *ptr ;
```

- The name class represents the address of the zero-th element of the structure array.
- ptr is a pointer to data objects of the type struct stud.

- **The assignment**

```
ptr = class ;
```

will assign the address of class[0] to ptr.

- **When the pointer ptr is incremented by one (ptr++) :**
  - The value of ptr is actually increased by sizeof(stud).
  - It is made to point to the next record.

- **Once ptr points to a structure variable, the members can be accessed as:**

```
ptr -> roll ;
ptr -> dept_code ;
ptr -> cgpa ;
```

  - The symbol "->" is called the arrow operator.

Example: Pointer

```
#include <stdio.h>

typedef struct {
            float real;
            float imag;
           } _COMPLEX;
```

```
swap_ref(_COMPLEX *a, _COMPLEX *b)
{
 _COMPLEX tmp;
  tmp=*a;
  *a=*b;
  *b=tmp;
}
```

```
print(_COMPLEX *a)
{
  printf("(%f,%f)\n",a->real,a->imag);
}
```

```
main()
{
 _COMPLEX x={10.0,3.0}, y={-20.0,4.0};

  print(&x); print(&y);
  swap_ref(&x,&y);
  print(&x); print(&y);
}
```

Things to remember

- **When using structure pointers, we should take care of operator precedence.**
  - **Member operator "." has higher precedence than "*".**
    - **ptr –> roll and (*ptr).roll mean the same thing.**
    - ***ptr.roll will lead to error.**
  - **The operator "–>" enjoys the highest priority among operators.**
    - **++ptr –> roll will increment roll, not ptr.**
    - **(++ptr) –> roll will do the intended thing.**

```
#include <stdio.h>
struct complex {
                    float re;
                    float im;
                  };

main()
{
   struct complex a, b, c;
   scanf ("%f %f", &a.re, &a.im);
   scanf ("%f %f", &b.re, &b.im);
   c = add (a, b) ;
   printf ("\n %f %f", c.re, c.im);
}
```

```
struct complex add (x, y)
struct complex x, y;
{
    struct complex t;

    t.re = x.re + y.re ;
    t.im = x.im + y.im ;
    return (t) ;
}
```

```
#include <stdio.h>
struct complex {
                    float re;
                    float im;
                  };

main()
{
   struct complex a, b, c;
   scanf ("%f %f", &a.re, &a.im);
   scanf ("%f %f", &b.re, &b.im);
   add (&a, &b, &c) ;
   printf ("\n %f %f", c,re, c.im);
}
```

```
void add (x, y, t)
struct complex *x, *y, *t;
{
    t->re = x->re + y->re ;
    t->im = x->im + y->im ;
}
```

Contd…

storage-class struct tag {

member 1 ;

member 2;

. . . . .

member m;

}variable 1, *variable 2,* . . ., *variable n;*

it is possible to combine the declaration of the structure composition with that of the structure variables

Contd…

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

oldcustomer and newcustomer are structure variables of type account

Contd…

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

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

Contd…

struct account customer

= {12345, ‘R’, “John W. Smith”, 586.30, 5, 24, 90};

//structure name account

//structure variable customer

Members: acct_no = 12345; acct_type= ‘R’; name[80] = “John W. Smith”; balance = 586.30; month = 5; day = 24; year = 90

Declaration: Structure Variable as an array

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

struct account
      {
int acct-no;
char acct-type;
char name[80];
float balance;
struct date lastpayment;
}    customer[ 100];
```

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

struct account {
int acct_no;
char acct-type;
char name[80];
float balance ;
struct date lastpayment;
} ;
struct account customer[100];
```

```
struct date {
char name[80];
int month;
int day;
int year;
}

struct date birthday[ ] = {“Amy”, 12, 30, 73, “Gail”, 5, 13, 66, “Marc”, 7, 15,
    72, “Marla”, 11, 29, 70,”Megan”, 2, 4, 77,”Sharonw”,12, 29, 63, “Susan”, 4,
    12,69};
```

```
#include<stdio.h>

struct studinfo{
     char name[80];
     char dept[30];
     int roll_no;
     };

main()
{
struct studinfo si;
printf("Your name:");
scanf(“ %[^\n]",si.name);
printf("\nYour department:");
scanf("%s",si.dept);
printf("\nYour roll number:");
scanf("%d",&si.roll_no);
printf("**************************");
printf("\nYour Name:%s",si.name);
printf("\nYour Department:%s",si.dept);
printf("\nYour Roll Number:%d",si.roll_no);
}
```

```
#include<stdio.h>

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

struct studinfo{
    char name[80];
    char dept[30];
    int roll_no;
    struct date birth;
    };
```

```
main( )
{
struct studinfo si;
printf("Your name:");
scanf(“ %[^\n]",si.name);
printf("\nYour department:");
scanf("%s",si.dept);
printf("\nYour roll number:");
scanf("%d",&si.roll_no);
printf("\nDate of birth:(mm/dd/yy)");
scanf("%d%d%d",&si.birth.month,&si.birth.day,&si.birth.year);
printf("*****************************");
printf("\nYour Name:%s",si.name);
printf("\nYour Department:%s",si.dept);
printf("\nYour Roll Number:%d",si.roll_no);
printf("\nYour Birthday:%d-%d-
%d",si.birth.month,si.birth.day,si.birth.year);
}
```

```
#include<stdio.h>

struct studinfo{
    char name[80];
    char dept[30];
    int roll_no;
    };

main()
{
int n,i;
struct studinfo si[100];
printf("How many student
    information are you storing: ");
scanf("%d",&n);
for(i=0;i<n;++i){
    printf("Your name:");
    scanf(" %[^\n]",si[i].name);
    printf("Your department:");
    scanf(" %[^\n]",si[i].dept);
    printf("Your roll number:");
    scanf("%d",&si[i].roll_no);
    }
printf("\n*****************************\n");
printf("*****************************");
printf("\n");
for(i=0;i<n;i++)
            {
     printf("\nYour Name:%s",si[i].name);
     printf("\nYour Department:%s",si[i].dept);
     printf("\nYour Roll Number:%d",si[i].roll_no);
printf("\n*****************************");
            }
}
```

```
#include<stdio.h>

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

struct studinfo{
    char name[80];
    char dept[30];
    int roll_no;
    struct date birth;
    };
```

```
main()
{
int n,i;
struct studinfo si[100];
printf("How many student information are you
    storing: ");
scanf("%d",&n);
for(i=0;i<n;++i){
    printf("Your name:");
    scanf(“ %[^\n]",si[i].name);
    printf("Your department:");
    fflush(stdin);
    scanf(“ %[^\n]",si[i].dept);
    printf("Your roll number:");
    scanf("%d",&si[i].roll_no);
    printf("\nDate of birth:(mm/dd/yy)");
    scanf("%d%d%d",&si[i].birth.month,&si[i].birth.d
    ay,&si[i].birth.year);
    }
```

```
printf("*****************************\n");
printf("*****************************");
printf("\n");
for(i=0;i<n;i++){
    printf("\nYour Name:%s",si[i].name);
    printf("\nYour Department:%s",si[i].dept);
    printf("\nYour Roll Number:%d",si[i].roll_no);
    printf("\nYour Birthday:%d-%d-
    %d",si[i].birth.month,si[i].birth.day,si[i].birth.year);
    printf("\n*********************************");
    }
}
```

typedef

- **typedef** feature allows users to define new data-types that are equivalent to existing data types

- new data type is defined **as**

  typedef ***type new- type;***

typedef int age;

age male, female;

is equivalent to writing

int male, female;

//Example

```
#include<stdio.h>
main()
{
typedef int sk;
sk a,b,sum;
printf("Enter two number");
scanf("%d%d",&a,&b);
sum=a+b;
printf("sum of %d and %d is
%d",a,b,sum);
}
```

Contd…

```
struct studinfo{
    char name[80];
    char dept[30];
    int roll_no;
    };
struct studinfo a,b;
```

```
typedef studinfo{
    char name[80];
    char dept[30];
    int roll_no;
    } studinfo;
studinfo a,*pa;
```

```
#include<stdio.h>

typedef struct{
      char name[80];
      char dept[30];
      int roll_no;
      }studinfo;

main()
{
studinfo a,*pa;
pa=&a;
printf("What is your name? ");
scanf(" %[^\n]",a.name);
printf("\n Your department ");
scanf(" %[^\n]",a.dept);
printf("\n Your Roll number ");
scanf("%d",&a.roll_no);
printf("\n**********************\n");
printf("Your Name:%s",pa->name);
printf("\nYour Department:%s",pa-
      >dept);
printf("\n Your Roll no:%d",pa->roll_no);
}
```

```
#include<stdio.h>

typedef struct{
      char name[80];
      char dept[30];
      int roll_no;
      }studinfo;

main()
{
studinfo a,*pa;
pa=&a;
printf("What is your name? ");
scanf(" %[^\n]",a.name);
printf("\n Your department ");
scanf(" %[^\n]",a.dept);
printf("\n Your Roll number ");
scanf("%d",&a.roll_no);
printf("\n**********************\n");
printf("Your Name:%s",(*pa).name);
printf("\nYour Department:%s",(*pa).dept);
printf("\n Your Roll no:%d",(*pa).roll_no);
}
```

```
#include<stdio.h>
typedef struct{
	char name[80];
	char dept[30];
	int roll_no;
	}studinfo;

main()
{
studinfo a,*pa;
pa=&a;
printf("What is your name? ");
scanf(" %[^\n]",a.name);
printf("\n Your department ");
scanf(" %[^\n]",a.dept);
printf("\n Your Roll number ");
scanf("%d",&a.roll_no);
printf("\n**********************\n");
printf("Your Name:%s",pa->name);
printf("\nYour Department:%s",pa->dept);
printf("\n Your Roll no:%d",pa->roll_no);
printf("\n**********************\n");
++pa->roll_no;
printf("Your Name:%s",pa->name);
printf("\nYour Department:%s",pa->dept);
printf("\n Your Roll no:%d",pa->roll_no);
}
```

```
#include<stdio.h>

int update(int n)
{
n=n+1;
return n;
}


typedef struct{
		char name[80];
		char dept[30];
		int roll_no;
		}studinfo;


main()
{
studinfo a,*pa;
pa=&a;
printf("What is your name? ");
scanf(" %[^\n]",a.name);
printf("\n Your department ");
scanf(" %[^\n]",a.dept);
printf("\n Your Roll number ");
scanf("%d",&a.roll_no);
printf("\n**********************\n");
printf("Your Name:%s",pa->name);
printf("\nYour Department:%s",pa->dept);
printf("\n Your Roll no:%d",pa->roll_no);
printf("\n**********************\n");
a.roll_no =update(a.roll_no);
printf("Your Name:%s",pa->name);
printf("\nYour Department:%s",pa->dept);
printf("\n Your Roll no:%d",pa->roll_no);
}
```

Unions

- Unions, like structures, contain members whose individual data types may differ from one another
- members within a union all share the same storage area within the computer's memory
- each member within a structure is assigned its own unique storage area

Contd…

- unions are used to conserve memory
- are useful for applications involving multiple members, where values need not be assigned to all of the members at any one time

Contd…

union *tag* {
member 1 ;
member 2;
. . . . .
member m;
};

storage-class union *tag* {
member 1 ;
member 2;
. . . . .
member m;
} variable 1 , variable 2, . . .,
variable n;

```
#include<stdio.h>

typedef union{
    char name[80];
    char dept[30];
    int roll_no;
    }studinfo;

main()
{
studinfo a;
printf("a store %d bytes",sizeof(a));
}
```

```
#include<stdio.h>

typedef union{
    char name[80];
    char dept[30];
    int roll_no;
    }studinfo;


main()
{
studinfo a;
printf("What is your name? ");
scanf(" %[^\n]",a.name);
printf("\n Your department ");
scanf(" %[^\n]",a.dept);
printf("\n Your Roll number ");
scanf("%d",&a.roll_no);
printf("\n**********************\n");
printf("Your Name:%s",a.name);
printf("\nYour Department:%s",a.dept);
printf("\n Your Roll no:%d",a.roll_no);
}
```