Structures and Unions

Structure Types

- Arrays are collections of related data of the same data type (homogeneous)
- Structures are user-defined collections of related data of possibly different data types (heterogeneous)
- There are different ways to define a structure

Structure Types (Cont.)

- One method:

struct struct_tag
{
$type_1$ id_list_1;
$type_2$ id_list_2;
...
};

KEYWORD

STRUCTURE TAG

STRUCTURE MEMBERS

Brackets delimit structure members

- Examples:

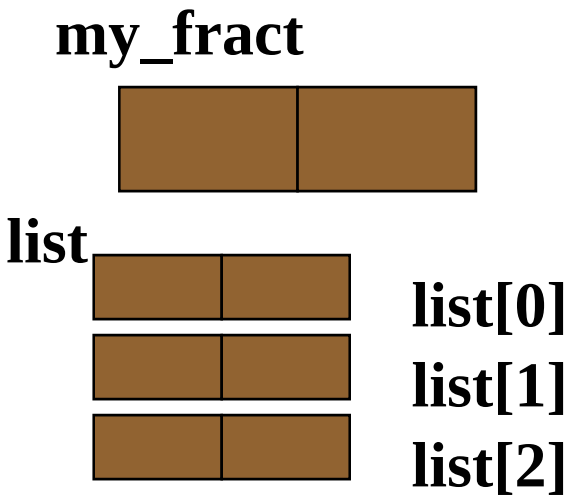
```
struct fract {
  int num, den;
};
```

```
struct personal {
  char  name[20];
  char  sex;
  int   age;
  float height;
};
```

Structure Types (cont.)

- No memory has been allocated to structure yet
- To declare variables of these structures:

```
struct fract   my_fract,list[3];
```




```
struct personal    aeron,group[20];
```

Structure Types (cont.)

- Second method: combine definition of a structure type and declaration of variables of that type together

```
struct fract {
  int num, den;
};
```

+

```
struct fract my_fract,
             list[3];
```

=

Tag is optional here.

```
struct fract {
  int num, den;
} my_fract, list[3];
```

Rules for declaring structure members

- Individual structure members may be of any common data types (int,float,etc..)
- All member names within a structure must be different although they can be similar to variables declared outside a structure
- Individual members cannot be initialized inside the structure declaration

struct basics

- Definition of a structure:

```
struct <struct-type>{
    <type> <identifier_list>;
    <type> <identifier_list>;
    ...
} ;
```

Each identifier defines a <u>member</u> of the structure.

- Example:

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

The "Date" structure has 3 members, day, month & year.

struct examples

- Example:

```
struct StudentInfo{
    int Id;
    int age;
    char Gender;
    double CGA;
};
```

The “StudentInfo” structure has 4 membe of different types.

- Example:

```
struct StudentGrade{
    char Name[15];
    char Course[9];
    int Lab[5];
    int Homework[3];
    int Exam[2];
};
```

The “StudentGrade” structure has 5 members of different array types.

struct examples

- Example:

```
struct BankAccount{
    char Name[15];
    int AcountNo[10];
    double balance;
    Date Birthday;
};
```

The “BankAcount” structure has simple, array and structure types as members.

- Example:

```
struct StudentRecord{
    char Name[15];
    int Id;
    char Dept[5];
    char Gender;
};
```

The “StudentRecord” structure has 4 members.

```
#include <stdio.h>
struct student
{ char name[50];
   int roll;
   float marks;
} s;
```

```
int main()
{
    printf("Enter info:\n");
    printf("Enter name: ");
    scanf("%s", s.name);
    printf("Enter roll number: ");
    scanf("%d", &s.roll);
    printf("Enter marks: ");
    scanf("%f", &s.marks);
    printf("Displaying Info:\n");
    printf("Name: ");
    puts(s.name);
    printf("Roll num: %d\n", s.roll);
    printf("Marks: %.1f\n", s.marks);
    return 0;
}
```

- Write a C program to add two distances entered by user. Measurement of distance should be in inch and feet.(Note: 12 inches = 1 foot)

```
#include <stdio.h>
struct Distance{
    int feet;
    float inch;
    }d1,d2,sum;
int main(){
    printf("1st distance\n");
    printf("Enter feet: ");
    scanf("%d",&d1.feet); /* input of feet for structure variable
    d1 */
    printf("Enter inch: ");
    scanf("%f",&d1.inch); /* input of inch for structure variable
    d1 */
```

```
printf("2nd distance\n Enter feet: ");
 scanf("%d",&d2.feet); /* input of feet for structure
   variable d2 */
printf("Enter inch: ");
scanf("%f",&d2.inch); /* input of inch for structure
   variable d2 */
sum.feet=d1.feet+d2.feet;
sum.inch=d1.inch+d2.inch;
if (sum.inch>12){ //If inch is greater than 12, changing
   it to feet.
 ++sum.feet; sum.inch=sum.inch-12; }
 printf("Sum of distances=%d\'%.1f\"",
   sum.feet,sum.inch); return 0; }
```

Using typedef

- For creating synonyms (aliases) for previously defined data types.
- Examples:

```
typedef int integ;
typedef char charac;
typedef struct card Card;
```

  - integ is now a synonym for type int; charac is a synonym for type char; and Card is a synonym for type struct card.
  - Now we can write:

```
integ a, b, c; /* a, b, c are of type int */
charac v;
```

Using typedef (cont.)

- Third method (most favoured), using typedef:

```
typedef struct {
  int num, den;
} Fract;

Fract my_fract, list[3];
```

  - Advantage: No need to mention ‘struct’ keyword while declaring structure variables now

Initializing Structures

- Similar to initialization of arrays. Example:

```
typedef struct {
int acct_no;     char name[20];     float balance;
} Customer;

    customer  a= { 12345,“marry", 566.50 };
```

  The above initializes the member acct_no of structure to 12345 ,name to marry and the member balance to 566.50 .

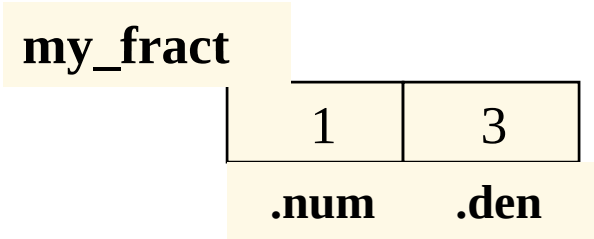
- Remaining members (if any) are initialized to 0 (or NULL for pointers).

Initializing Structures (cont.)

- What does this code do?

```
typedef struct {
  int num, den;
} Fract;

Fract my_fract = { 1, 3 };
```



Accessing Members of Structures

- Use the structure member operator (also called the dot operator) (.) to access a member of a structure variable.
- Examples:

```
Fract my_fract = { 1, 3 };
printf("Numerator is %d\n", my_fract.num);
printf("Denominator is %d\n", my_fract.den);

Custmor a = {12345, “marry", 566.50 };
printf(“acct_ni is  %d\n", a.acct_no);
printf(“name is %s\n", a.name);
Printf(“balance is %f”,a.balance);
```

Accessing Members of Structures (cont.)

- As is the case for arrays, you cannot compare two structure variables by simply comparing their names. You need to compare their individual members.
- Example:

```
int A[3] = { 2, 4, 6 };
int B[3] = { 2, 4, 6 };

if (A == B)   /* this is wrong! */
  printf("A and B are the same.\n");
```

Accessing Members of Structures (cont.)

- Example:

```
Fract f1 = { 1, 3 };
Fract f2 = { 1, 3 };

/* this is wrong! */
if (f1 == f2)
  printf("f1 and f2 are the same.\n");

/* this is correct */
if (f1.num == f2.num && f1.den == f2.den)
  printf("f1 and f2 are the same.\n");
```

Example:

```
main()
{
struct {
int integer;
float real;
} first_structure,second_structure;
first_structure.integer=7;
first_structure.real=3.14;
second_structure.integer=first_structure.integer;
second_structure.real=first_structure.real;
printf("integer=%d, real=%fd",
    second_structure.integer,second_structure.real);
}
```

Output:

Integer=7 , real =3.14

Example: Planets

- We want to store information on planets.

  Name: Jupiter
  Diameter: 142,800 km
  Moons: 4
  Orbit time: 11.9 years
  Rotation time: 9.925 hours

- Declare a structure type planet_t:

```
#define STRSIZE 10
typedef struct
{
    char  name[STRSIZE];
    float diameter;
    int   moons;
    float orbit_time, rotation_time;
} planet_t ;
```

Array of structures

- Structure is used to store the information of One particular object but if we need to store such 100 objects then Array of Structure is used.

```
struct Bookinfo
{
 char bname[20];
int pages;
 int price;
 } Book[100];
```

```
#include <stdio.h>
 struct Bookinfo
{
 char[20] bname;
 int pages;
int price;
 }book[3];

int main(int argc, char *argv[])
 {   int i;
      for(i=0;i<3;i++)
  {  printf("\nEnter the Name of Book : ");
     gets(book[i].bname);
     printf("\nEnter the Number of Pages : ");
     scanf("%d",book[i].pages);
     printf("\nEnter the Price of Book : ");
     scanf("%f",book[i].price);
}
   printf("\n--------- Book Details -------
    ----- ");

  for(i=0;i<3;i++)
{  printf("\nName of Book :
    %s",book[i].bname);
    printf("\nNumber of Pages :
    %d",book[i].pages);
    printf("\nPrice of Book :
    %f",book[i].price);
    }
return 0;  }
```

Nesting of structures

- Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure.

Option 1

```
struct date
{    int date;
     int month;
     int year;
};
struct Employee
 { char ename[20];
     int ssn;
     float salary;
     struct date doj;
}emp={"Pritesh",1000,1000.50,{22,
     6,1990}};;
```

```
int main(int argc, char *argv[])
     {
     printf("\nEmployee Name :
     %s",emp.ename);
      printf("\nEmployee SSN :
     %d",emp.ssn);
     printf("\nEmployee Salary :
     %f",emp.salary);
     printf("\nEmployee DOJ :
     %d/%d/%d", \
     emp.doj.date,emp.doj.month,emp.
     doj.year);
     return 0;
      }
```

Option 1

```
#include <stdio.h>
struct Employee
{    char ename[20];
     int ssn;
     float salary;
     struct date
     { int date;
       int month;
       int year;
      }doj; } emp =
     {"Pritesh",1000,1000.50,{22,6,19
     90}};
```

```
int main(int argc, char *argv[])
     {
     printf("\nEmployee Name :
     %s",emp.ename);
      printf("\nEmployee SSN :
     %d",emp.ssn);
     printf("\nEmployee Salary :
     %f",emp.salary);
     printf("\nEmployee DOJ :
     %d/%d/%d", \
     emp.doj.date,emp.doj.month,emp.doj
     .year);
     return 0;
      }
```

```c
#include <stdio.h>
#include <string.h>

struct student_college_detail
{
    int college_id;
    char college_name[50];
};

struct student_detail
{
    int id;
    char name[20];
    float percentage;
    // structure within structure
    struct student_college_detail
     clg_data;
}stu_data;

int main()
{   struct student_detail stu_data = {1,
     "Raju", 90.5, 71145,"AnnaUniversity"};
  printf(" Id is: %d \n", stu_data.id);
  printf(" Name is: %s",stu_data.name);
  printf(" Per is: %f \n",
    stu_data.percentage);

  printf(" College Id : %d ",
                stu_data.clg_data.college_id
    );
   printf(" College Name is: %s \n",
                stu_data.clg_data.college_n
    ame);
   return 0; }
```

Example: Planets (cont.)

- Declare variables of this `planet_t` type:

```
planet_t    this_planet,  prev_planet,
            blank_planet = {"", 0.0, 0, 0.0, 0.0};
```

Variable **blank_planet:**

name	`\0 ? ? ? ? ? ? ? ? ?`
diameter	`0.0`
moons	`0`
orbit_time	`0.0`
rotation time	`0.0`

Example: Planets (cont.)

- We can build hierarchical structures
- Example of a ‘solar system’ structure:

```
typedef struct {
 float     diameter;
 planet_t  planets[9];
 char      galaxy[STRSIZE];
} solar_sys_t;
```

ARRAY OF STRUCTURES: 9 STRUCTURE VARIABLES OF ‘PLANET’ TYPE

  - Note the use of planets structure inside solar_sys_t structure

Example: Planets (cont.)

- Manipulating individual components of a structure:

```
strcpy (this_planet.name, "Jupiter");
this_planet.diameter = 142800;
this_planet.moons = 4;
this_planet.orbit_time = 11.9;
this_planet.rotation_time = 9.925;
```

- More examples:

```
solar_sys_t      solar;
solar.diameter = 2.3e20;          /*not real figure*/
strcpy (solar.planets[2].name, "Jupiter");
solar.planets[2].diameter = 142800;

/* and so on */
```

Example: Planets (cont.)

- Manipulating whole structure: the name of a structure type variable refers to the entire structure
- The statement below copies the values of all the components of **this_planet** into the corresponding components of **prev_planet**:

```
prev_planet = this_planet;
```

- This is different from arrays, as this is disallowed for arrays

Example: Planets (cont.)

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

typedef struct {
char  name[10];
float diameter;
int   moons;
float orbit_time, rotation_time;
} planet_t ;

typedef struct {
float     diameter;
 planet_t  planets[9];
 char      galaxy[10];
} solar_sys_t;
```

```
void main()
{
planet_t    this_planet,prev_planet;
clrscr();
strcpy (this_planet.name, "Jupiter");
 this_planet.diameter = 142800;
 this_planet.moons = 4;
 this_planet.orbit_time = 11.9;
this_planet.rotation_time = 9.95;
solar_sys_t       solar;
            solar.diameter = 2.3e20;  /*not real figure*/
            strcpy (solar.planets[2].name, "Jupiter");
           solar.planets[2].diameter = 142800;
prev_planet = this_planet;
printf("diameter is %f\n",this_planet.diameter);
printf("moons are %d\n",this_planet.moons);
printf("orbit time  is %f\n",this_planet.orbit_time);
printf("rotation time%f\n",this_planet.rotation_time);
printf("diameter is %f\n",prev_planet.diameter);
getch();          }
```

Assigning the contents of one structure variable to another structure variable of the same type:

```
struct s_type  {
   int a;
   float f;
}  var1, var2;
var1.a = 10;
var1.f = 100.23;

var2 = var1;
```

- After this fragment executes, **var2** will contain exactly the same thing as **var1**

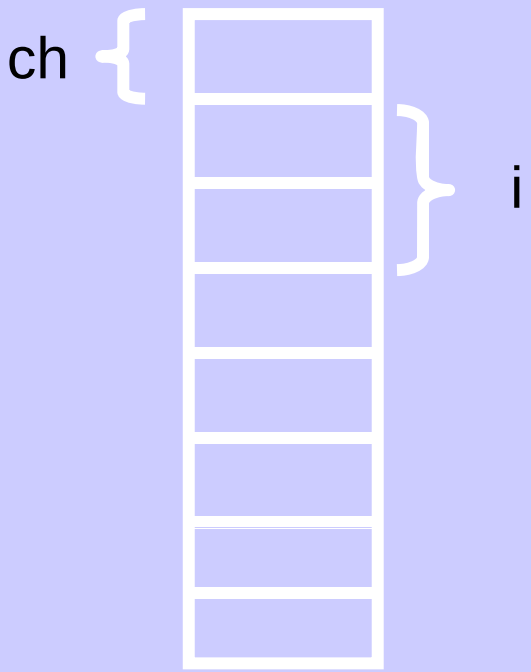
Union Types

- Union like structure, contains members whose individual data type may differ from one another
- However, the members within a union all share the same storage area within the computer memory
- The union data type was invented to prevent memory fragmentation and build heterogeneous data structures
- Declaration example:

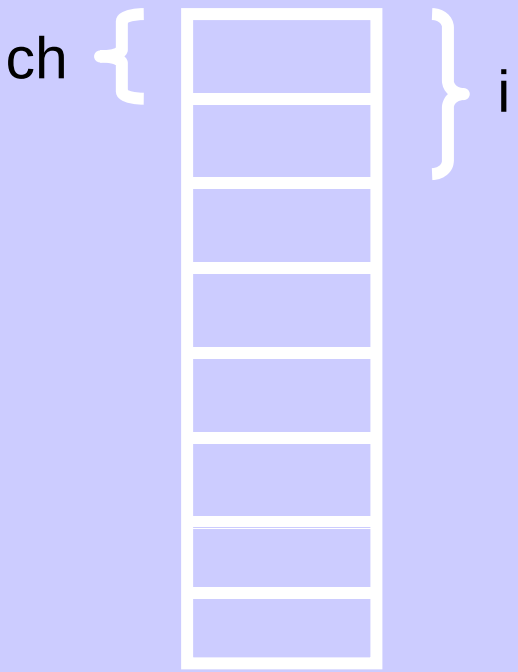
```
union id
{
  int   size;
  char  color[12];
} shirt;
```

Structures vs. Unions (1)

```
struct number{
    char ch;
    int i;
} x;
```




```
union number{
    char ch;
    int i;
}  y;
```



Structures vs. Unions (2)

- Members of a union can only be accessed one at a time

```
x.i = 100;
x.ch = 'a';
printf("%c,%d",x.ch,x.i);
```

Output:

a,100

```
y.i = 100;
y.ch = 'a';
printf("%c,%d",y.ch,y.i);
```

Output:

a,1627390052

- Since union members have the same memory address, changing the value of one alters the value of the other

```c
/* Fig. 10.5: fig10_05.c
   An example of a union */
#include <stdio.h>

/* number union definition */
union number {
   int x;
   double y;
}; /* end union number */

int main( void )
{
   union number value; /* define union variable */

   value.x = 100; /* put an integer into the union */
   printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n\n\n",
      "Put a value in the integer member",
      "and print both members.",
      "int:", value.x,
      "double:", value.y );

```

Fig. 10.5 | Displaying the value of a union in both member data types. (Part 1 of 2.)

```c
    value.y = 100.0; /* put a double into the same union */
    printf( "%s\n%s\n%s\n  %d\n\n%s\n  %f\n",
       "Put a value in the floating member",
       "and print both members.",
       "int:", value.x,
       "double:", value.y );
    return 0; /* indicates successful termination */
 } /* end main */
```

```
Put a value in the integer member
and print both members.
int:
  100

double:
  -9255959211743313600000000000000000000000000000000000000000000000.000000


Put a value in the floating member
and print both members.
int:
  0

double:
  100.000000
```

Fig. 10.5 | Displaying the value of a union in both member data types. (Part 2 of 2.)

Example:

A union may be member of a structure, and a structure may be a member of a union

```
union id {
char color[12];
int size;
};
struct clothes{
char manufacturer[20];
float cost;
union id description;
} shirt;
```

Combined code →

```
struct clothes{
char manufacturer[20];
float cost;
union {
char color[20];
int size;
}description;
} shirt;
```

Why Do We Need Unions?

- Suppose there are three types of products: books, mugs, and shirts.
  - Books: Title, author, number of pages
  - Mugs: Design
  - Shirts: color, size

- We want to store all the items in an array. So, we must use the same data type to describe all three different kinds of products

Structure Approach

```
struct catalog_item{
    int stock_number;
    float price;
    int item_type;
    char title[100];
    char author[100];
    int num_pages;
    char design[100];
    int color;
    int size;
}
```

```
int j;
struct catalog_item x[5];
x[0].item_type = 0;
strcpy(x[0].title,"unix");
……
for(j = 0; j<5;j++){
 switch(x[j].item_type)
{
case 0: puts(x[j].title); break;
case1: puts(x[j].design); break;
case2: ……
 }
}
```

Use Unions to Save Memory (1)

```
struct item{
    int stock_number;
    float price;
    int item_type;
    union{
      struct{
        char title[100];
        char author[100];
        int num_pages;
      } book;
      struct{
        char design[100];
      } mug;
      struct {
        int size;
        char color[20];
      } shirt;
    } item;
};
```

Use Unions to Save Memory (2)

```
int j;
struct item x[5];
x[0].item_type = 0;
strcpy(x[0].book.title,"unix");
……
for(j = 0; j<5;j++)
{
 switch(x[j].item_type)
{
    case 0: puts(x[j].book.title); break;
    case 1: puts(x[j].mug.design); break;
    case 2: ……
 }
}
```