GE6151 COMPUTER PROGRAMMING
LECTURE 9

STRUCTURES AND UNIONS

Prof. Dr. M. Paulraj PhD.,
SRIT
Coimbatore-10

- Structures are collection of related variables.
- Structures are derived data types.
- Structures may contain variables of many different data types. Thus a single structure might contain integer elements, floating point elements, character elements.
- The individual structure elements are known as members.

The general syntax of structure declaration is:

```
struct name_tag {
        member_1;
        member_2;
        member_3;
        …
        member_n;
};
```

where “struct” is a keyword, name_tag is a name that identifies the structure.
The individual members can be ordinary variables, pointers, arrays and other structures.

The individual structure type variables can be declared as:

struct name_tag var1, var2,…, varn;

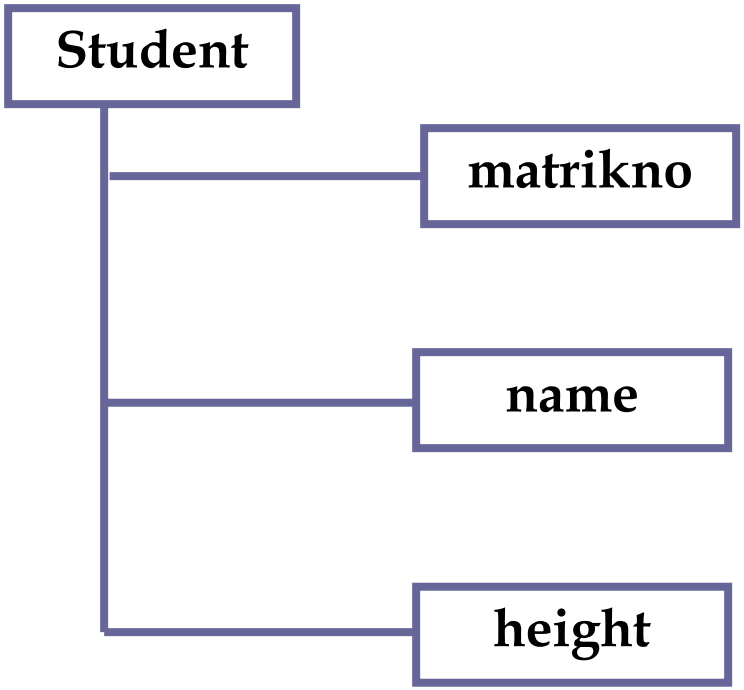
The variables var1, var2,.., varn are structure variables of type name_tag.

Example 1:

```
struct student{
        int matrikno;
        char name[40];
        float height;
};
```

In the above example, the name of the structure is "student" and the structure members are matrikno, name and height.

The composition of the structure is shown in the following figure.



To declare the structure variables first and second the following statement can be made:

```
struct student  first, second;
```

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

```
struct name_tag {
        member_1;
        member_2;
        member_3;
        …
        member_n;
} var1, var2, var3,…, varn;
```

Note: The name_tag is optional in this case.

Example 2:

```
struct student{
                        int matrikno;
                        char name[40];
                        float height;
                } first, second;
```

first and second are structure variables
of type student.

Example 3:

```
struct stationery {
        int pencil;
        float pencilcost;
        int pen;
        float pencost;
} royal ;
```

```
struct date {
        int day;
        int month;
        int year
} date_arrival, date_departure;
```

```
struct book {
        int accno;
        char title[250];
        char author[40];
        int copies
} cbook, cplusbook;
```

```
struct complex {
        float real;
        float imag;
} a,b;
```

Example 4:
An array of structure can be defined as :

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

```
struct account {
        int accountno;
        char name[50];
        int accounttype;
        float balance;
        struct date lastpayment;
} customer[100];
```

Note: the last member in the structure account is of type structure date which has been defined earlier.

The array declaration can be done separately as:

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

struct account {
        int accountno;
        char name[50];
        int accounttype;
        float balance;
        struct date lastpayment;
};

struct account customer[100];
```

Note: Each array element is a structure consisting of five members

namely accountno, name, accounttype, balance and lastpayment.

Assignment of initial values to the structure can be made.

Example 5:

```
struct student {
                    int matno;
                    char name[50];
                    float height;
                    };

struct student s1 ={ 78104, “Rahim”,60.5);
```

Example 6:

```
struct studate{
                    int matno;
                    int month;
                    int date;
                    int year;
};

struct studate  birthday = {
                    1234,10,21,1991,
                    };
```

Scope rule of a structure:

The name of a member of a structure can appear as a name in another structure.

Example 7:

```
struct structure1{
        int a;
        float b; };

struct structure2{
        int b;
        float a; };
```

How the member of a structure variable can be accessed?

A structure member can be accessed by a dot operator. The syntax for using a dot operator is :

```
Variable.member
```

where variable refers to the name of a structure type variable and member represents to the name of a member within a structure.

Example 8:

```
struct studentdate{
                    int matno;
                    int month;
                    int date;
                    int year;
} joiningdate;
```

The members matno,month,date,year of the structure variable joiningdate can be accessed by writing

```
joiningdate.matno
joiningdate.month
joiningdate.date
joiningdate.year
```

Note : The dot operator has the precedence level of 17.

Several expressions involving structure variables are:

++joiningdate.matno
joingdate.date++
&joingdate
&joiningdate.year
Complex expressions involving repeated use of dot operators can be used.

Example 9:

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

```
struct studentdate{
        int matno;
        strcut date d1;} feepaiddate;
```

To access the month of the feepaiddate we can use feepaiddate.d1.month and this value can be processed by other operators.

The use of period operator can be extended to array of structures.

Example 10:

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

```
struct account {
        int accountno;
        char name[50];
        int accounttype;
        float balance;
        struct date lastpayment;
} customer[100];
```

The month of last payment made by the second customer is:

customer[1].lastpayment.month

User defined data types:

The user can create new data types that are equivalent to existing data types using typedef.

The syntax of typedef statement is:
typedef type new-type;

where typedef is the key word, type represents existing data type and new-type represents the new data type name.

Example 11:

```
typedef int code;
code kt1813,kz1063;
```

Example 12:

```
typedef char string[60];
string name;
```

The above statements are equivalent to defining

```
char name[60];
```

Example 13:

```
typedef struct{
        int matno;
        int name[50];
        float midmark;
        float labmark;
        float quizmark;
        float finalmark;
} record;

record student1, student2;
record student[100];
```

STRUCTURES AND POINTERS

Pointers can be used to point structure variables.

Example 14:

```
struct smark{
                int matno;
                char name[50];
                int mark;
                }  s, *student1;
```

student1 is a pointer variable of type structure smark.

The individual member of the structure can be accessed using the **structure selection operator ->.**

```
student1 -> matno
student1 -> mark
student1 -> name[2]
```

Example 15:

```
typedef struct{
int  month;
int day;
int year;
} date;
                        struct {
                                int acc_no;
                                char acc_type;
                                char name[50];
                                float balance;
                                date lastpayment;
                        } customer, *pcus = &customer;
```

customer.acc_no
pcus -> acc_no
(*pcus).acc_no

customer.balance
pcus -> balance
(*pcus).balance

Passing structures to a function

Individual structure members can be passed to a function as an argument.

```
#include<stdio.h>
float interest(float prin,
float rate, int year);
void main(void)  {
typedef struct {
float principal;
float rate;
float inter;
int nyear;
} loan;
loan acc;
acc.principal = 39000.0;
acc.rate = 3.9;
acc.nyear = 7;
acc.inter=interest(acc.principal,acc.rate
,acc.nyear);
printf("The interst amount is %f\n",
acc.inter);
return;
}
float interest(float prin, float rate, int
year)
{
float temp;
return(prin*rate*(float) year/100.0);
}
```

Result:

The interest amount is 10647.0000

Example 16:

```
#include<stdio.h>
typedef struct{
        float real;
        float imag;
} rect;
void change(rect *v);

void main(void)  {
rect v1;
v1.real = 3.0;
v1.imag = 4.0;
printf("The value before change is \n");
printf("%f %f\n",v1.real,v1.imag);
change(&v1);
printf("The value after
change is \n");
printf("%f
%f\n",v1.real,v1.imag);
return;  }

void change(rect *v){
 v->real = 4.0;
 v->imag = 3.0;
 return;  }
```

The value before change is 3.0000 4.0000
The value after change is 4.0000 3.0000

UNION

The Union is a construct that allows memory to be shared by different types of data. The syntax of union is similar to that of a structure.

The syntax of union is:

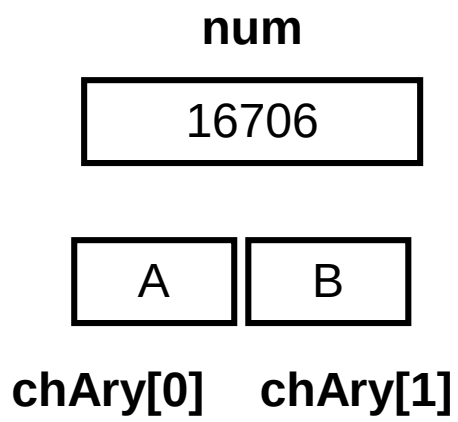
```
union tagname{
member_1;
member_2;
…
member_n;
} variable_1, variable_2,…,variable_n;
```

The members that compose a union share the storage area within the computer's memory.

Example

```
/* short int size is 2 bytes */
typedef  union
    {
      short num;
     char chAry[2];
    } SH_CH2;

SH_CH2 data;
```



Example

```
#include<stdio.h>

typedef  union
    {
      short num;
     char chAry[2];
    } SH_CH2;

int main (void)
{
SH_CH2 data;
data.num = 16706;
printf("Short %hd\n", data.num);
printf("ch[0] %c\n",data.chAry[0]);
printf("ch[0] %c\n",data.chAry[1]);
return 0;
}
```

Results:

```
Short 16706
Ch[0] A
Ch[1] B
```