UNIT 7

STRUCTURE & UNION

Compiled by: Er. Krishna Khadka

INTRODUCTION

- Structure is collection of heterogeneous data items treated as a single unit.
- It is a mechanism for packing data of different types.
- It is a convenient tool for handling a group of logically related data items. For example, it can be used to represent a set of attributes, such as student_name, roll_no and marks.
- Each data item is included in a structure is called ‘member’ of structure.
- It helps to organize complex data item in a more meaningful way.
- The keyword ‘struct’ is used to declare structure variable and type of structure variable is defined by the ‘tag_name’ of the structure.

MAJOR FEATURES OF STRUCTURE

- It can be treated as record that is collection of interrelated data fields having different data types.
- It is possible to copy one structure variable to another by simply assignment operator (=).
- It allows nested structure that is one structure inside another structure.
- It is possible to pass structure variable as parameter to any function.
- It is possible to define structure pointer also known as linked list

STRUCTURE CONTD..

- A structure creates a data type that can be used to group items of possibly different types into a single type.

How to create a structure?

- struct keyword is used to create a structure.

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memeberN;
};
```

Example:

```
struct employee
{   int id;
    char name[20];
    float salary;
};
```

STRUCTURE DECLARATION

```
struct tag_name
{
        data_type member 1;
        data_type member 2;
        ……………………
        data_type member n;
};
struct tag_name var 1, var 2,var 3….var m;
                    "OR"
struct tag_name
{
        data_type member1;
        data_type member2;
        ……………………
        data_type memberN;
}var1, var2,var3….varM;
```

STRUCTURE DECLARATION

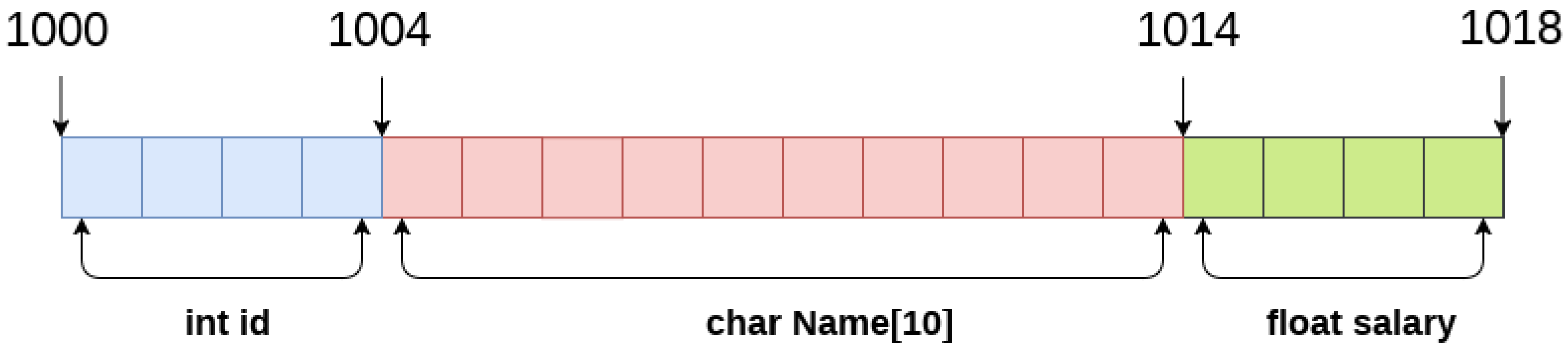
- In this declaration ‘struct’ is a keyword, ‘tag_name’ is a name of structure that identifies structure of this type, member1, member2,……membern are individual member declarations.
- The individual members can be ordinary variables, arrays, pointers or other structures.
- The member names within a structure must be distinct from one another.

STRUCTURE DECLARATION CONTD.

- For example:

```
 struct student
{
          int roll_no;
          char fname[30];
          char lname[30];
};
struct student s1,s2,s3;
                    "OR"
struct student
{
          int roll_no;
          char fname[30];
          char lname[30];
}s1,s2,s3;
```

The following image shows the memory allocation of the structure employee that is defined in the above example.

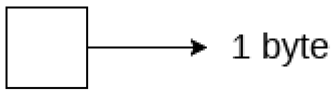


```
struct Employee
{
      int id;
      char Name[10];
      float salary;
} emp;
```

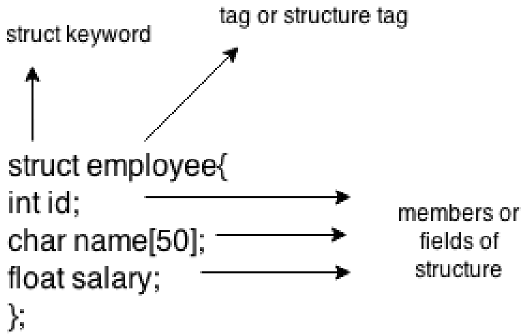
sizeof (emp) = 4 + 10 + 4 = 18 bytes

where;

sizeof (int) = 4 byte
sizeof (char) = 1 byte
sizeof (float) = 4 byte



Here, **struct** is the keyword; **employee** is the name of the structure; **id**, **name**, and **salary** are the members or fields of the structure. Let's understand it by the diagram given below:



```
struct employee{
int id;
char name[50];
float salary;
};
```

STRUCTURE INITIALIZATION

- Structure variable can be initialized quite similar to array. The following example shows the structure initialization process.

```
struct student
{
        int roll_no;
        char fname[30];
        char lname[30];
}       s1={1, “Ram” ,“Rai”};
```

- The alternate method to initialize structure variable as:

```
struct student s2={2, “Sita” ,“Devi”};
struct student s3={3, “Hari” ,“Sharma”};
```

RULES FOR STRUCTURE INITIALIZATION

- We cannot initialize individual members inside the structure template.
- The order of values enclosed in braces must match the order of members in the structure definition.
- It is permitted to have a partial initialization. We can initialize only the first few members and leave the remaining blank. The uninitialized members should be only at the end of the list.
- The uninitialized members will be assigned default values as follows:
  a. zero for integer and floating point numbers.
  b. ‘\0’ for characters and strings

ACCESSING STRUCTURE MEMBERS

- A member of structure can be accessed by using period (.) sign between structure variable and respective member.

  Syntax

  variable.member

  Example

  s1.roll_no   s2.fname   s3.lname

ACCESSING STRUCTURE MEMBERS

- There are two other ways of accessing structure members.
- struct sample

```
struct sample
{
        int x;
        float y;
};
struct sample v, *ptr;
ptr=&v;
```

Here ‘ptr’ is a pointer that has been assigned the address of the structure variable ‘v’. Now, the members can be accessed in three ways.

using dot notation:	v.x
using indirection notation:	(*ptr).x
using selection notation:	ptr->x

EXAMPLE

```
#include<stdio.h>
struct student
{   int roll;
    char fname[30];
    char lname[30];
};
void main()
{   struct student s;
    printf("\nEnter roll no =");
    scanf("%d",&s.roll);
    printf("\nEnter First Name =");
    scanf("%s",s.fname);
    printf("\nEnter Last Name =");
    scanf("%s",s.lname);
    printf("Roll No = %d \nFirst Name = %s \nLast Name =
    %s",s.roll, s.fname,s.lname);
}
```

ARRAY OF STRUCTURE

- The collection of structure variables having same set of members with same identifier is called array of structure.
- **Example:**

```
struct student
{       int roll;
        char fname[30];
        char lname[30];
};
struct student s[5];
```

INITIALIZATION OF ARRAY OF STRUCTURE

- A structure can be initialized in the same way as that of array element in C.
- For example:

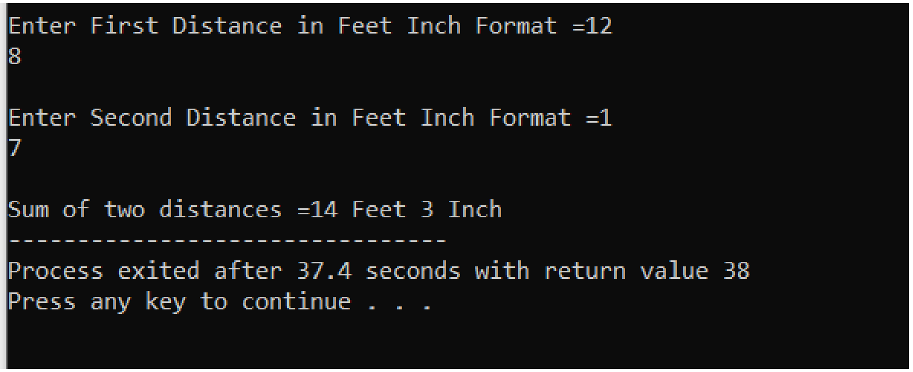
```
        struct school
        {
                int roll;
                char name;
                float weight;
        };
        struct school s[2]=  {
                                    {1, “Raju”, 72.4},
                                    {2, “Kaju”, 92.6}
                             };
```

A C program that takes roll no, first name & last name of 5 students and prints the same records on screen.

```
#include<stdio.h>
struct student
{   int roll;
    char fname[20];
    char lname[20];
}   s[5];
void main()
{   int i;
    for(i=0; i<5; i++)
    {           printf("\nEnter Roll Number =");
                scanf("%d",&s[i].roll);
                printf("\nEnter First Name =");
                scanf("%s",s[i].fname);
                printf("\nEnter Last Name =");
                scanf("%s",s[i].lname);
    }
    for(i=0;i<5;i++)
    printf("\n%5d %5s%5s",s[i].roll,s[i].fname,s[i].lname);
}
```

A PROGRAM TO CALCULATE SUM OF TWO DISTANCES AND DISTANCE IS MEASURED IN TERMS OF FEET AND INCH.

```
#include<stdio.h>
struct distance
{   int feet;
    int inch;
};
void main()
{
    struct distance d1,d2,d;
    printf("\nEnter First Distance in Feet Inch Format =");
    scanf("%d%d",&d1.feet,&d1.inch);
    printf("\nEnter Second Distance in Feet Inch Format =");
    scanf("%d%d",&d2.feet,&d2.inch);
    d.inch=(d1.inch+d2.inch)%12;
    d.feet=d1.feet+d2.feet+(d1.inch+d2.inch)/12;
    printf("\nSum of two distances =%d Feet %d Inch
    ",d.feet,d.inch);
}
```



```
Enter First Distance in Feet Inch Format =12
8

Enter Second Distance in Feet Inch Format =1
7

Sum of two distances =14 Feet 3 Inch
--------------------------------
Process exited after 37.4 seconds with return value 38
Press any key to continue . . .
```

NESTED STRUCTURE

- A structure can be defined as a member of another structure. This is known as nested structure.
- In the following example, the structure date is defined within structure student.
- **Example**

```
struct date
{        int day;
         int month;
         int year;
};
struct student
{        int roll_no;
         char fname[30];
         char lname[30];
         struct date DOB;       //date of birth
};
```

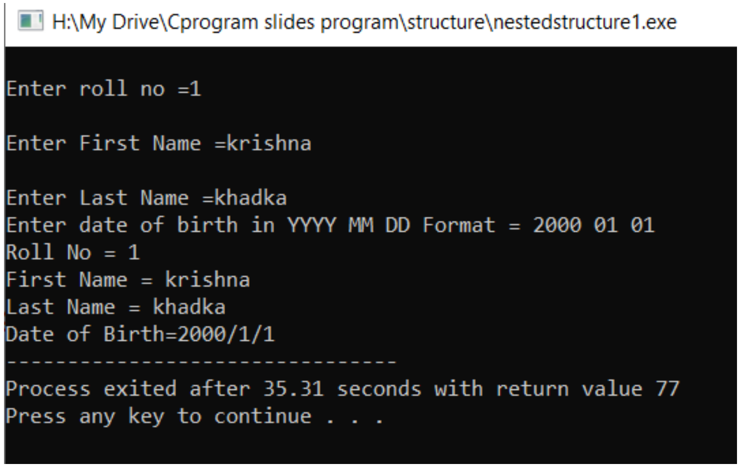
NESTED STRUCTURE CONTD.

- **Example**

```
struct student
{     int roll_no;
      char fname[30];
      char lname[30];
      struct date
      {       int day;
              int month;
              int year;
      }
      DOB;  //date of birth
}S;
```

A PROGRAM THAT TAKES ROLL NO, FIRST NAME, LAST NAME AND DATE OF BIRTH AND DISPLAYS THE INFORMATION ON SCREEN.

```
#include<stdio.h>
struct date
{           int year;
            int month;
            int day;     };
struct student
{           int roll_no;
            char fname[30];
            char lname[30];
            struct date d;              };
void main()
{           struct student s;
            printf("\nEnter roll no =");
            scanf("%d",&s.roll_no);
            printf("\nEnter First Name =");
            scanf("%s",s.fname);
            printf("\nEnter Last Name =");
            scanf("%s",s.lname);
            printf("Enter date of birth in YYYY MM DD Format = ");
            scanf("%d%d%d",&s.d.year,&s.d.month,&s.d.day);
            printf("Roll No = %d \nFirst Name = %s \nLast Name = %s \nDate of Birth
            =%d/%d/%d",s.roll_no,s.fname,s.lname,s.d.year,s.d.month,s.d.day);
}
```



H:\My Drive\Cprogram slides program\structure\nestedstructure1.exe

```

Enter roll no =1

Enter First Name =krishna

Enter Last Name =khadka
Enter date of birth in YYYY MM DD Format = 2000 01 01
Roll No = 1
First Name = krishna
Last Name = khadka
Date of Birth=2000/1/1
--------------------------------
Process exited after 35.31 seconds with return value 77
Press any key to continue . . .
```

A PROGRAM WHICH READS ROLL NO, FIRST NAME AND LAST NAME OF 5 STUDENTS AND PRINTS THE SAME RECORDS IN ASCENDING ORDER ON THE BASIS OF ROLL NO.

```
#include<stdio.h>
struct student
{   int roll;
    char fname[20];
    char lname[20];
}   s[5];
void main()
{   struct student temp;
    int i,j;
    for(i=0; i<5; i++)
    {       printf(“\nEnter Roll Number:");
            scanf("%d",&s[i].roll);
            printf("\nEnter First Name:");
            scanf("%s",s[i].fname);
            printf("\nEnter Last Name:");
            scanf("%s",s[i].lname);
    }
```

CONTD.

```
for(i=0;i<4;i++)
  {     for(j=i+1;j<5;j++)
        {       if(s[i].roll>s[j].roll)
                {       temp=s[i];
                        s[i]=s[j];
                        s[j]=temp;
                }
        }
  }
  for(i=0;i<5;i++)
        printf("\n%10d %10s %10s"
  ,s[i].roll,s[i].fname,s[i].lname);
}
```

ARRAYS WITHIN STRUCTURE

- C allows the use of arrays as structure members.
- Example:

```
struct marks
{
        int number;
        float subject[3];
}student[2];
```

Where the subject contains three elements, subject[0], subject[1], subject[2];

SIZE OF STRUCTURE

- The actual memory size of a variable in terms of bytes may vary from machine to machine. Therefore, the **sizeof** operator helps us to determine memory size of a variable.
- **Syntax:** sizeof(struct variable);
- **Example:**

```
struct student
{           int roll_no;
            char fname[30];
            char lname[30];

}S;

sizeof(struct S);
```

- In the above example, the memory size of the structure variable ‘s’ is 62.

  roll_no : 2Bytes/ 4 Bytes , depends(here we considered 2)

  fname: 30 Bytes

  roll_no: 30 Bytes

PASSING STRUCTURE TO FUNCTION

- Structure can be passed into function as arguments like other data types such as integer, floating point value, array etc.
- There are three methods by which the values of a structure can be transferred from one function to another.
  1. Passing structure members to function
  2. Passing entire structure to function
  3. Passing structure pointer to function

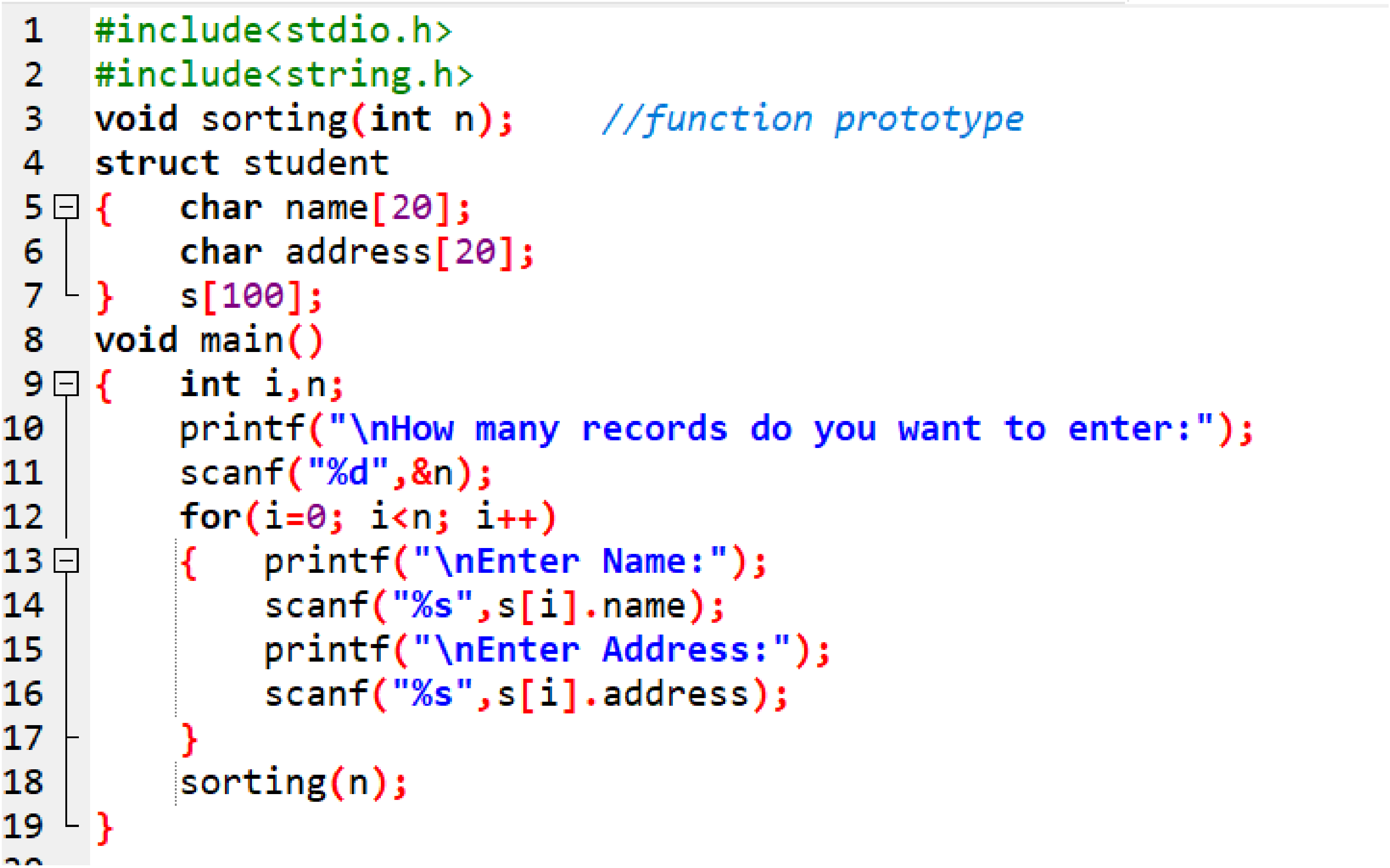
PASSING STRUCTURE MEMBER TO FUNCTION

- Individual structure 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.
- The actual arguments are then treated independently like ordinary variables.
- This is the most elementary method becomes unmanageable and insufficient when the structure size is large.

EXAMPLE

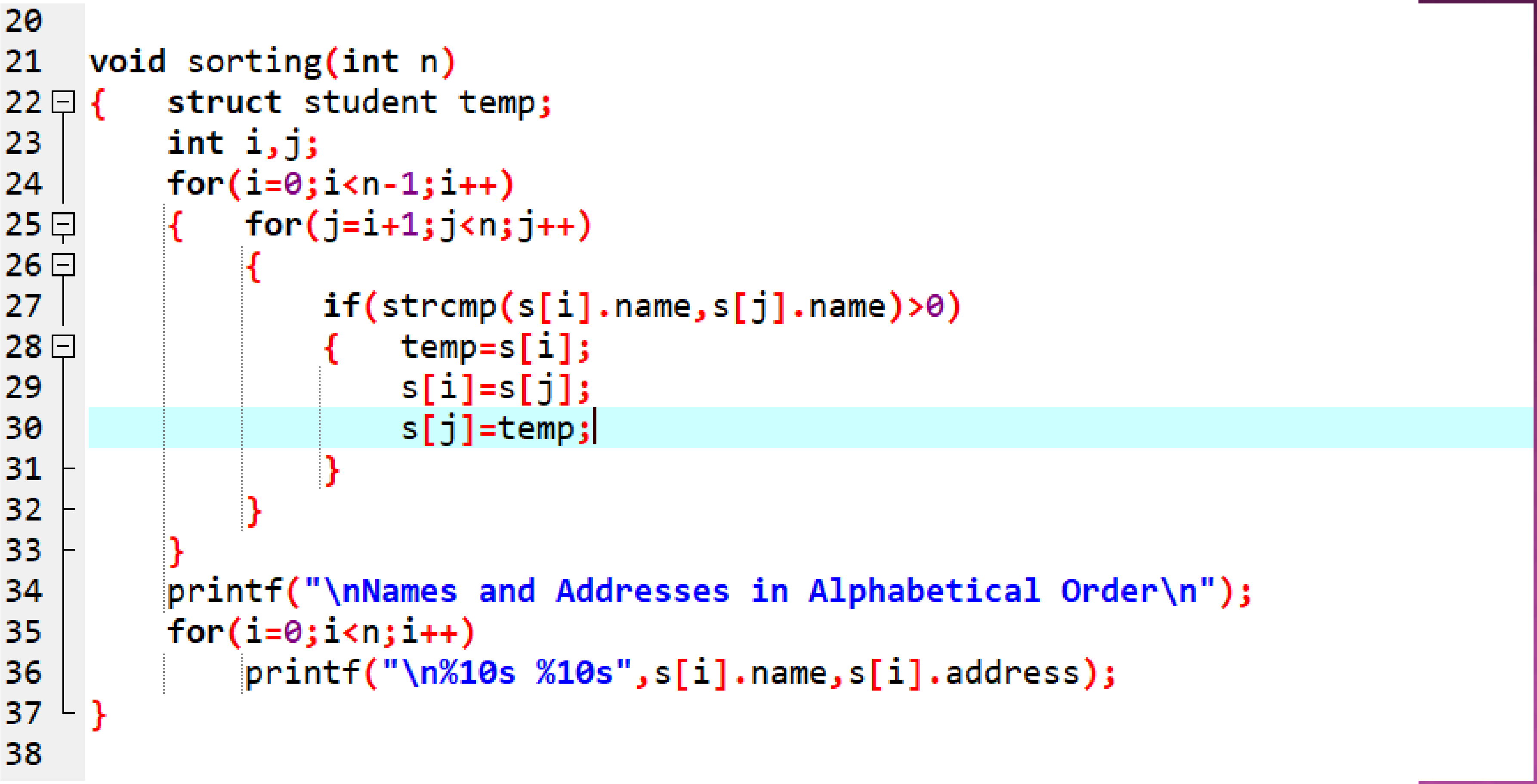
```
#include<stdio.h>
void display(int, float);
int main()
{       struct sample
        {
                int x;
                float y;
        };
        struct sample s1={1,2};
        display(s1.x,s1.y);
        return 0;
}
void display(int a, float b)
{
        printf("%d\t%f",a,b);
}
```

A PROGRAM THAT READS NAMES AND ADDRESS OF DIFFERENT STUDENTS AND REARRANGES THEM ON THE BASIS OF NAME IN ALPHABETICAL ORDER.

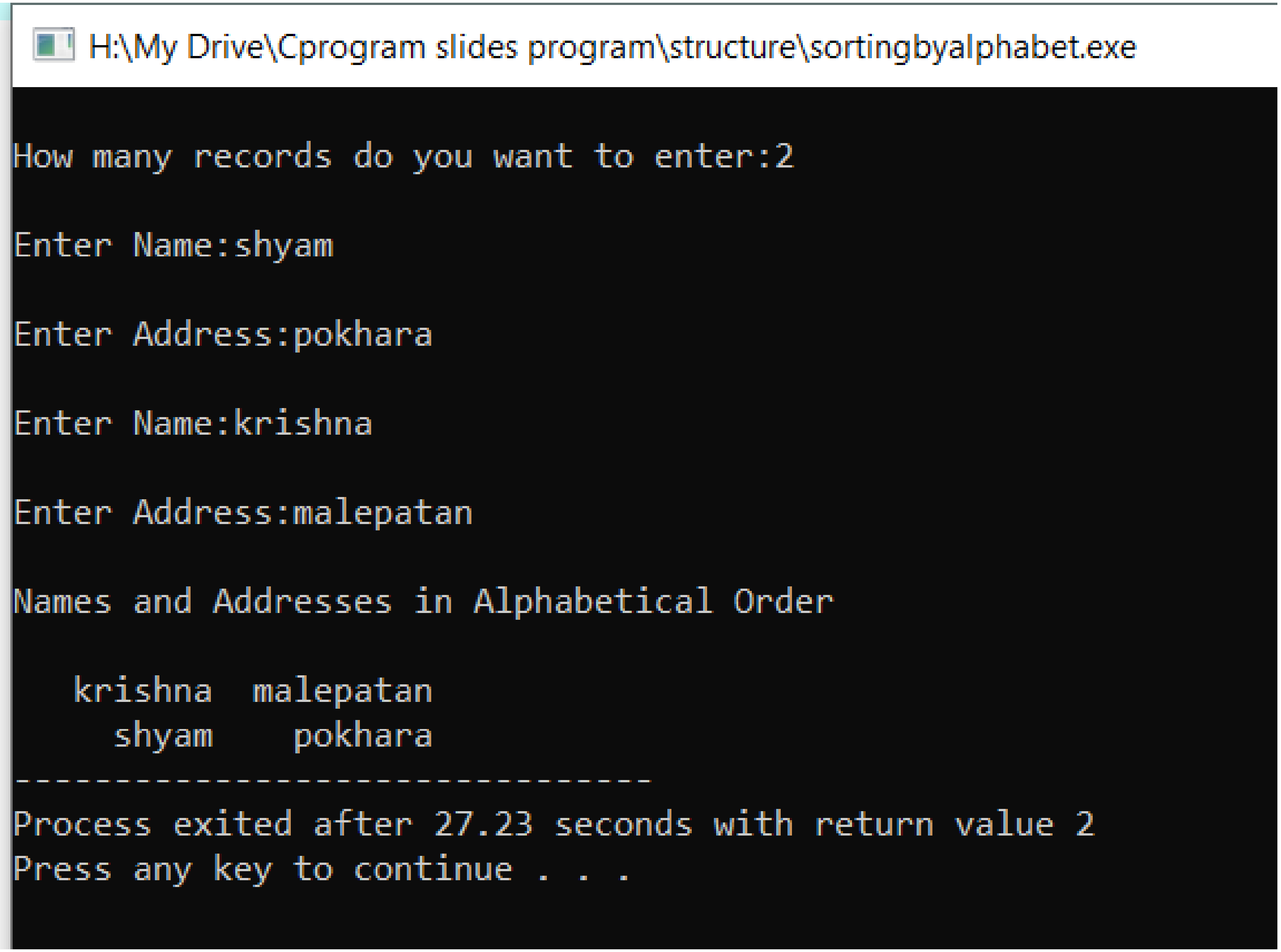


```c
#include<stdio.h>
#include<string.h>
void sorting(int n);    //function prototype
struct student
{    char name[20];
     char address[20];
}    s[100];
void main()
{    int i,n;
     printf("\nHow many records do you want to enter:");
     scanf("%d",&n);
     for(i=0; i<n; i++)
     {    printf("\nEnter Name:");
          scanf("%s",s[i].name);
          printf("\nEnter Address:");
          scanf("%s",s[i].address);
     }
     sorting(n);
}
```

CONTD.



```

void sorting(int n)
{   struct student temp;
    int i,j;
    for(i=0;i<n-1;i++)
    {   for(j=i+1;j<n;j++)
        {
            if(strcmp(s[i].name,s[j].name)>0)
            {   temp=s[i];
                s[i]=s[j];
                s[j]=temp;
            }
        }
    }
    printf("\nNames and Addresses in Alphabetical Order\n");
    for(i=0;i<n;i++)
        printf("\n%10s %10s",s[i].name,s[i].address);
}

```



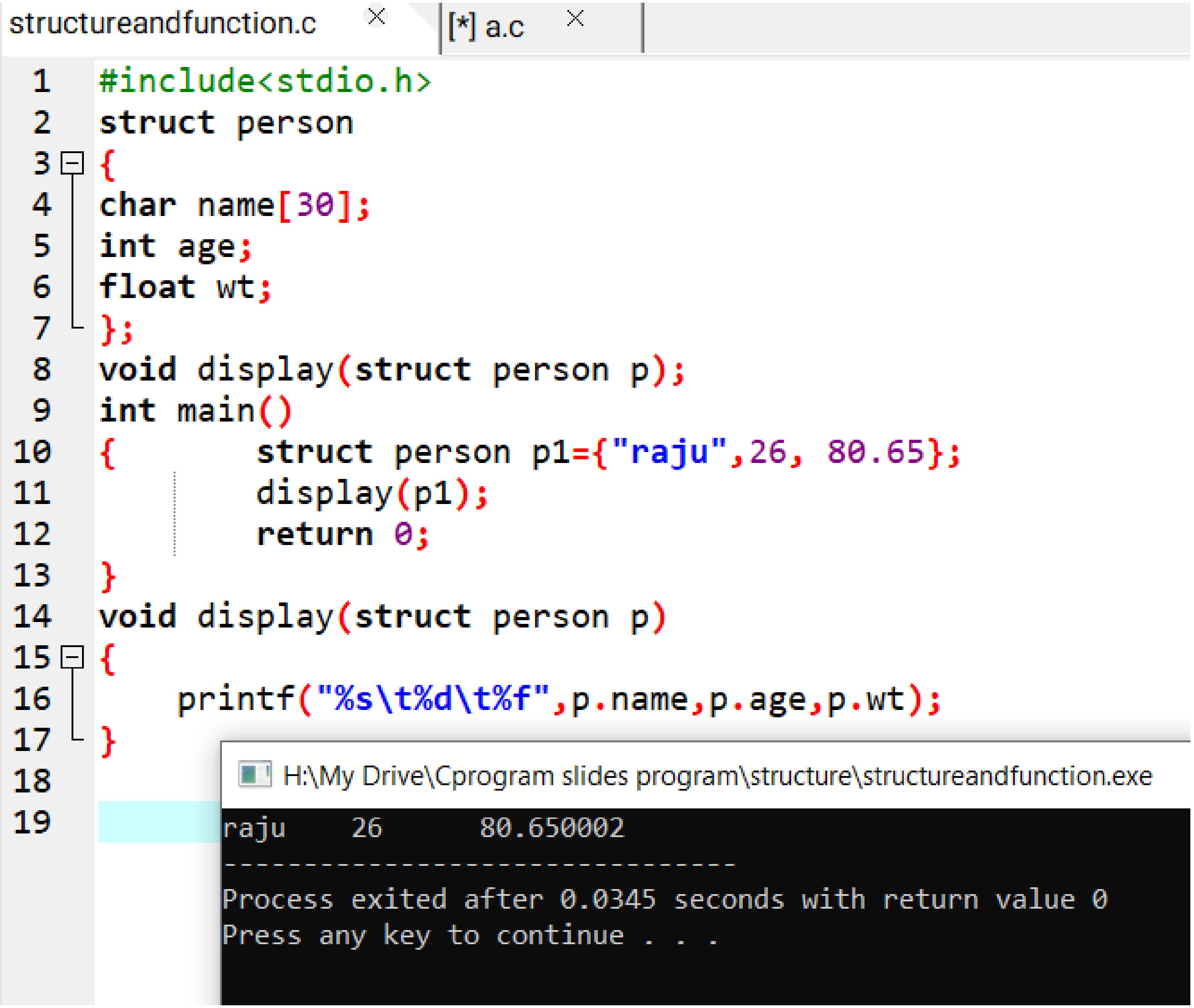
H:\My Drive\Cprogram slides program\structure\sortingbyalphabet.exe

```
How many records do you want to enter:2

Enter Name:shyam

Enter Address:pokhara

Enter Name:krishna

Enter Address:malepatan

Names and Addresses in Alphabetical Order

   krishna  malepatan
     shyam    pokhara
--------------------------------
Process exited after 27.23 seconds with return value 2
Press any key to continue . . .
```

PASSING STRUCTURE VARIABLE TO FUNCTION

- It involves passing of a copy of the entire structure to the called structure.
- Since the function is working on a copy of the structure, any changes to structure members within the function are not reflected in the original structure(in the calling function).

EXAMPLE


structureandfunction.c
[*] a.c
H:\My Drive\Cprogram slides program\structure\structureandfunction.exe

```c
#include<stdio.h>
struct person
{
char name[30];
int age;
float wt;
};
void display(struct person p);
int main()
{        struct person p1={"raju",26, 80.65};
         display(p1);
         return 0;
}
void display(struct person p)
{
    printf("%s\t%d\t%f",p.name,p.age,p.wt);
}


```

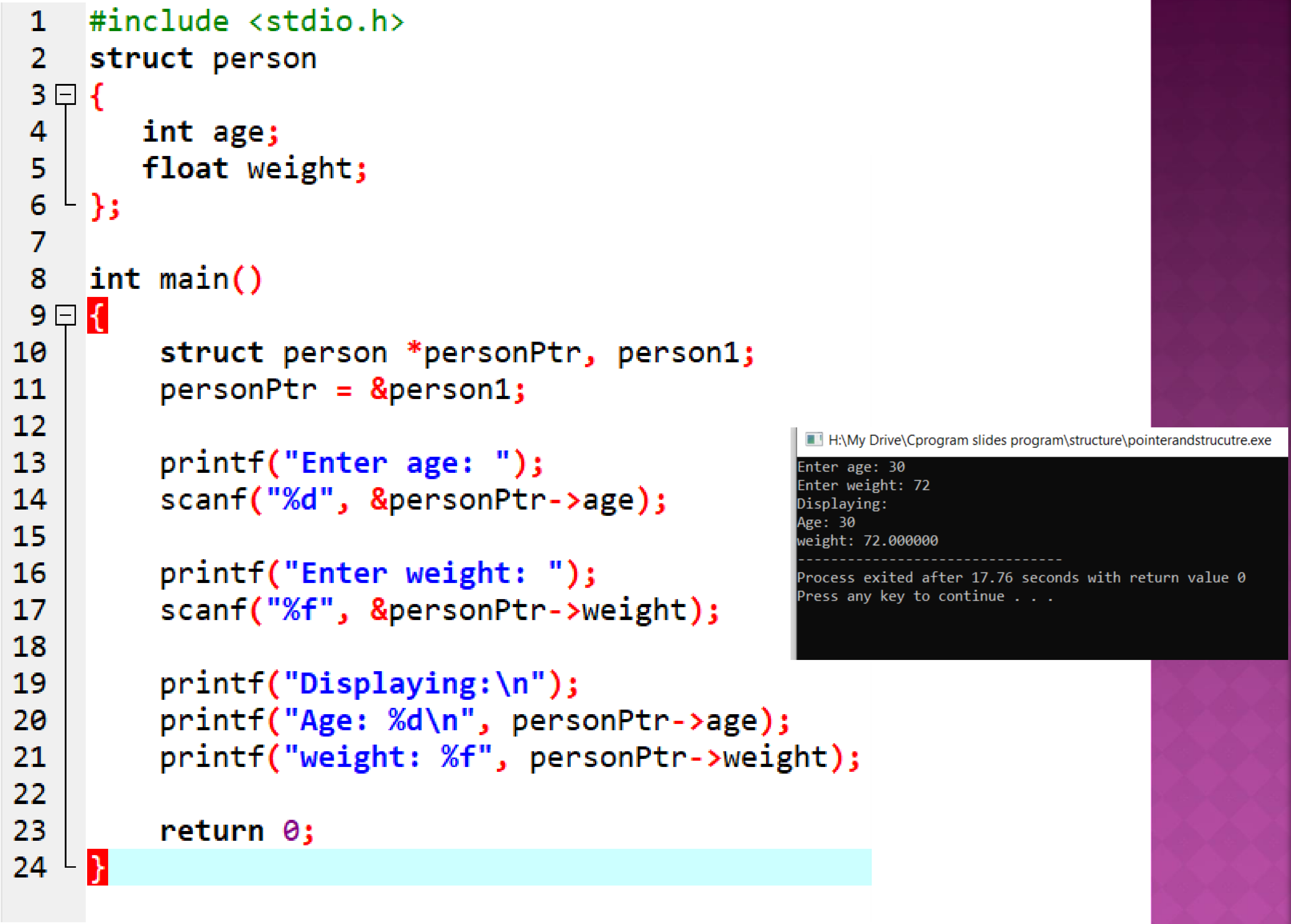
```
raju    26      80.650002
--------------------------------
Process exited after 0.0345 seconds with return value 0
Press any key to continue . . .
```

POINTER TO STRUCTURE

- C allows pointer to structure just as it allows pointer to any other type of variable. Like other pointers, structure pointers are declared by placing * in front of the structure variable name. For example:

```
struct sample
{
        int acc_no;
        char acct_type;
        char name[80];
        float balance;
};
struct sample customer, *ptr;
```

In this example ‘customer’ is a structure variable of type ‘sample’ and ‘ptr’ is a pointer variable whose object is a structure variable of type ‘sample’



```c
#include <stdio.h>
struct person
{
    int age;
    float weight;
};

int main()
{
    struct person *personPtr, person1;
    personPtr = &person1;

    printf("Enter age: ");
    scanf("%d", &personPtr->age);

    printf("Enter weight: ");
    scanf("%f", &personPtr->weight);

    printf("Displaying:\n");
    printf("Age: %d\n", personPtr->age);
    printf("weight: %f", personPtr->weight);

    return 0;
}
```

H:\My Drive\Cprogram slides program\structure\pointerandstrucutre.exe

```
Enter age: 30
Enter weight: 72
Displaying:
Age: 30
weight: 72.000000
--------------------------------
Process exited after 17.76 seconds with return value 0
Press any key to continue . . .
```

PASSING STRUCTURE POINTER TO FUNCTION

- An individual structure member can be accessed in terms of its corresponding pointer variable by writing

  ptvar->member

  where ‘ptvar’ refers to a structure type pointer variable and the -> is comparable to the period(.) operator.
- The expression **ptvar->member** is equivalent to **variable.member** where variable is a structure type variable.
- The -> operator can be combined with period operator to access a sub-member within a structure. Hence a sub-member can be accessed by writing

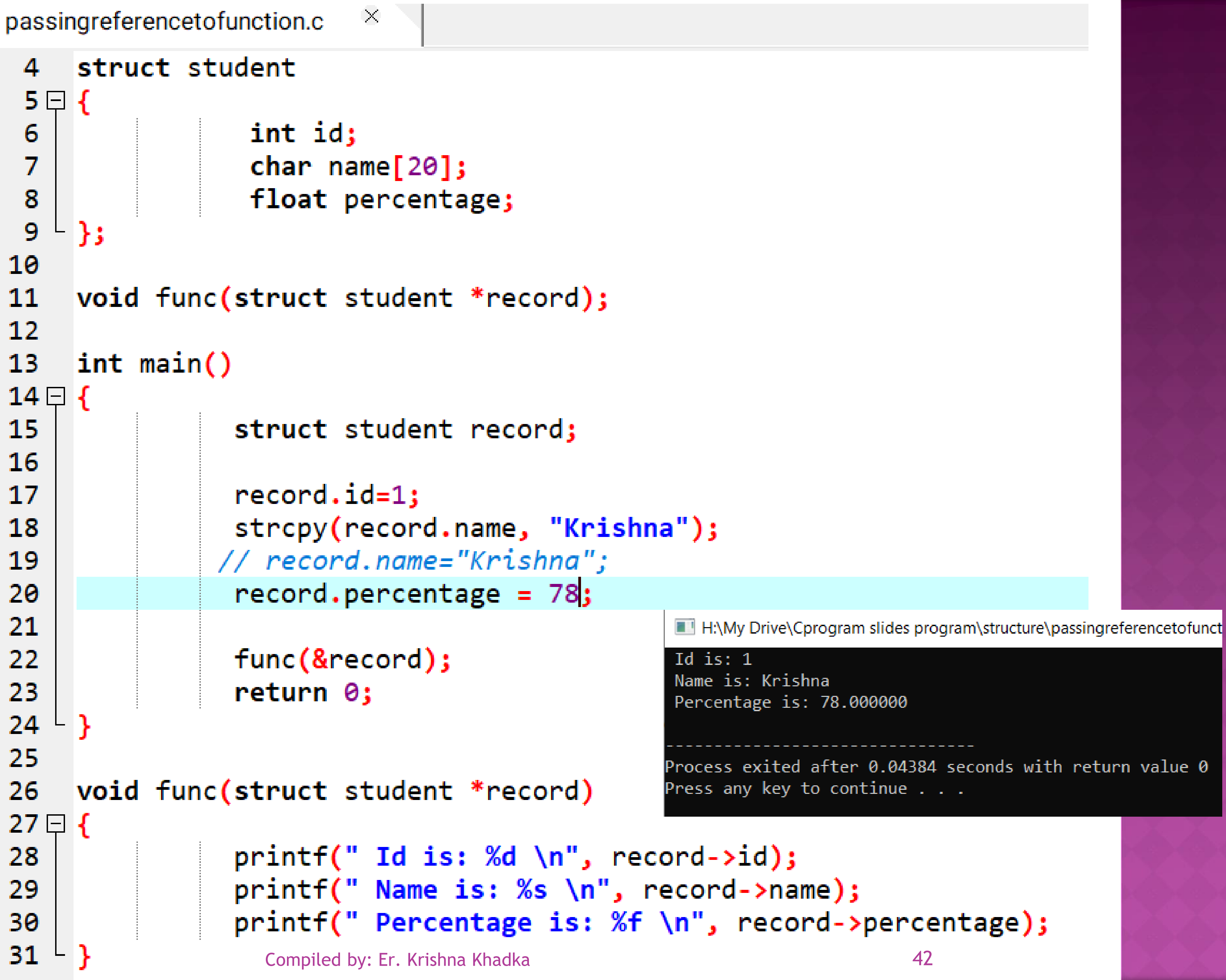
  ptvar->member.submember
- Similarly, the -> operator can be used to access an element of an array that is a member of a structure. This is done by writing

  ptvar->member[expression]

  Where expression is a nonnegative integer that indicates the array element.

PASSING STRUCTURE POINTER TO FUNCTION

- When a structure pointer is passed to a function, only the address of a structure variable is passed to the function.
- This makes very fast function calls.
- Passing a pointer makes it possible for the function to modify the contents of the structure used as the argument.



passingreferencetofunction.c

```c
struct student
{
        int id;
        char name[20];
        float percentage;
};

void func(struct student *record);

int main()
{
       struct student record;

       record.id=1;
       strcpy(record.name, "Krishna");
      // record.name="Krishna";
       record.percentage = 78;

       func(&record);
       return 0;
}

void func(struct student *record)
{
       printf(" Id is: %d \n", record->id);
       printf(" Name is: %s \n", record->name);
       printf(" Percentage is: %f \n", record->percentage);
}
```

H:\My Drive\Cprogram slides program\structure\passingreferencetofunct

```
 Id is: 1
 Name is: Krishna
 Percentage is: 78.000000

--------------------------------
Process exited after 0.04384 seconds with return value 0
Press any key to continue . . .
```

UNION

- Union is similar to structure but it differs only in its storage location.
- The union keyword is used to define union.
- In structure, each member has its own memory block whereas all members of union can share the same memory location.
- Therefore, union can take less amount of memory than structure and it can access only one member at a time.

Syntax for the declaration of union

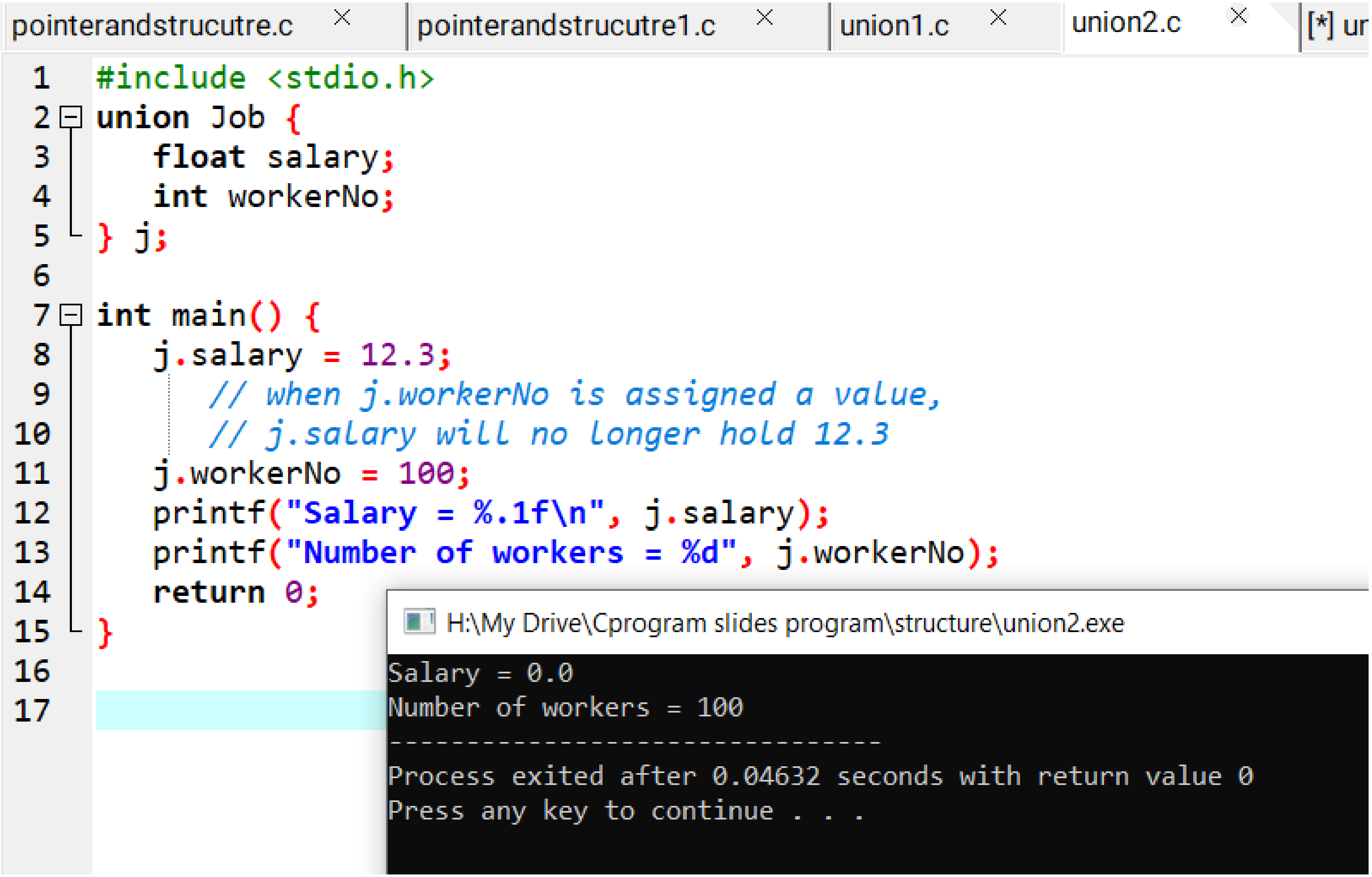
```
union tag_name
{  data_type member1;
   data_type member2;
   ..........................
   data_type membern;
};
union tag_name var1, var2,var3;
```

where member1, member2,……membern are the members of the union and var1, var2 and var3 are variables with data type tag_name.

EXAMPLE

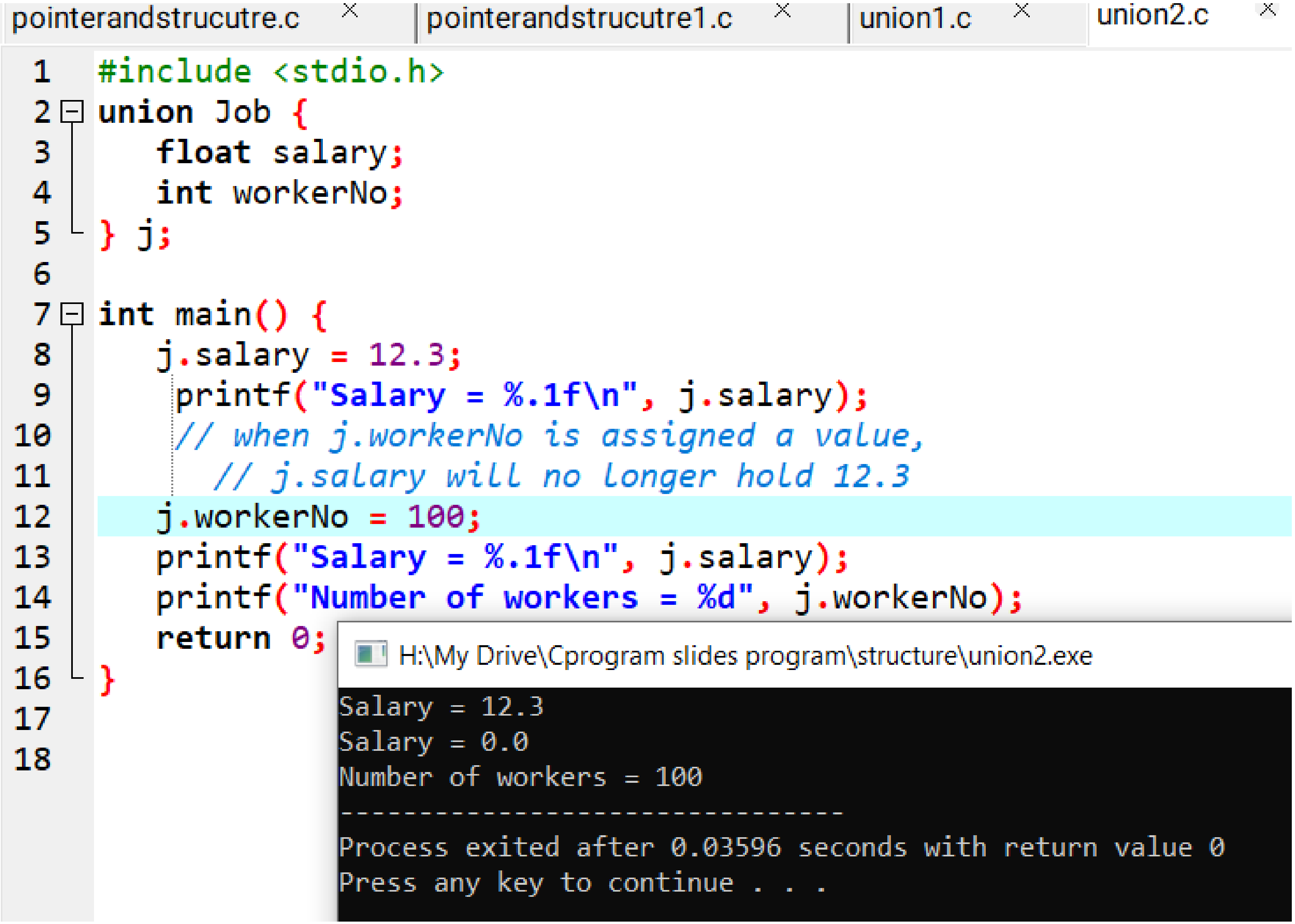
```
union sample
{
  char x;
  int y;
  float z;
} var;
```

- In this example, var is union variable with data type sample.
- It contains three members: char x, int y and float z.
- In this example, float z is the largest size member that requires 4 bytes memory and the same memory is shared over other two members.



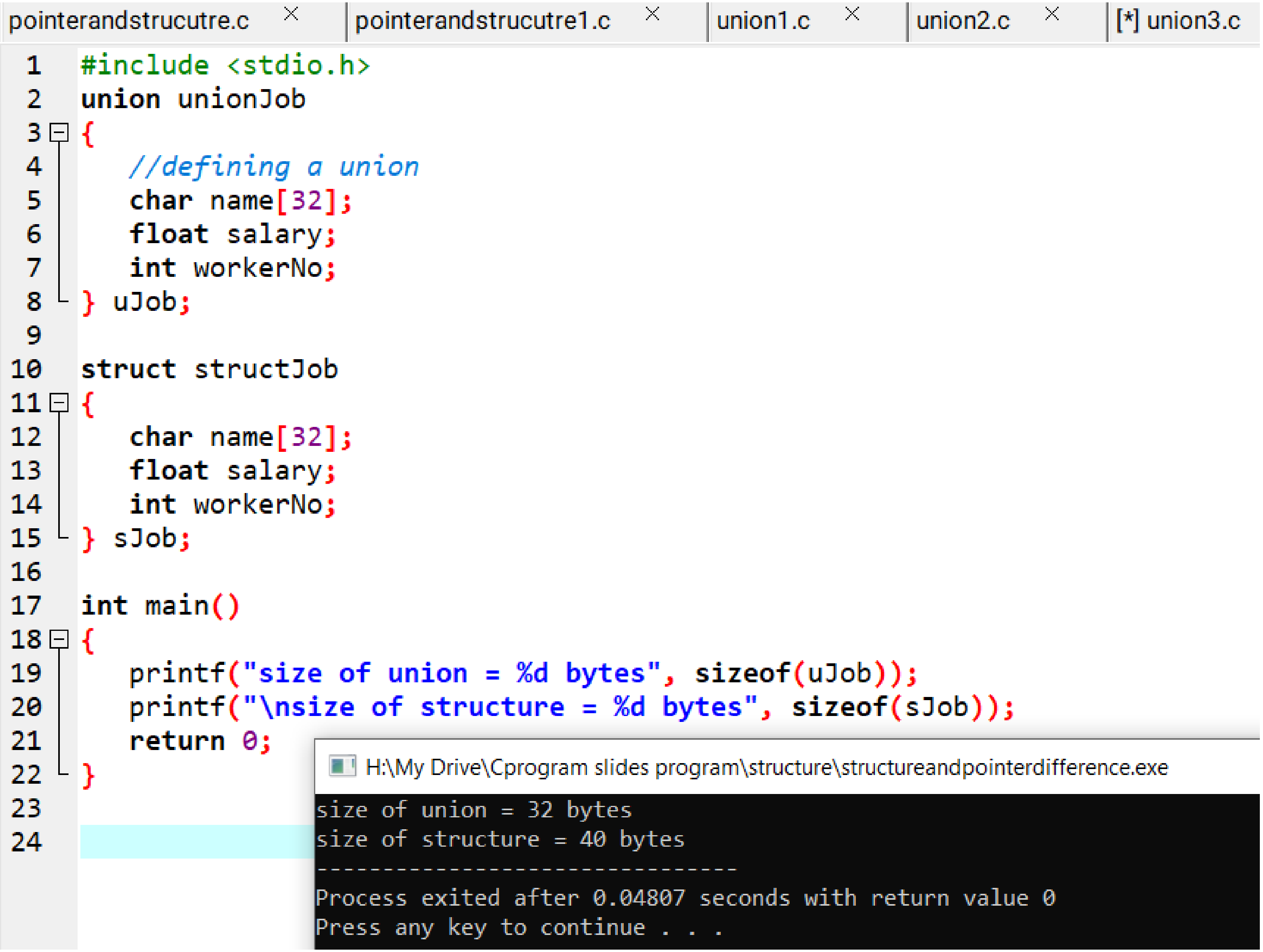
```c
#include <stdio.h>
union Job {
    float salary;
    int workerNo;
} j;

int main() {
    j.salary = 12.3;
      // when j.workerNo is assigned a value,
      // j.salary will no longer hold 12.3
    j.workerNo = 100;
    printf("Salary = %.1f\n", j.salary);
    printf("Number of workers = %d", j.workerNo);
    return 0;
}
```

```
Salary = 0.0
Number of workers = 100
--------------------------------
Process exited after 0.04632 seconds with return value 0
Press any key to continue . . .
```



```c
#include <stdio.h>
union Job {
   float salary;
   int workerNo;
} j;

int main() {
   j.salary = 12.3;
    printf("Salary = %.1f\n", j.salary);
    // when j.workerNo is assigned a value,
      // j.salary will no longer hold 12.3
   j.workerNo = 100;
   printf("Salary = %.1f\n", j.salary);
   printf("Number of workers = %d", j.workerNo);
   return 0;
}
```

```
Salary = 12.3
Salary = 0.0
Number of workers = 100
--------------------------------
Process exited after 0.03596 seconds with return value 0
Press any key to continue . . .
```


pointerandstrucutre.c
pointerandstrucutre1.c
union1.c
union2.c
[*] union3.c
H:\My Drive\Cprogram slides program\structure\structureandpointerdifference.exe

```c
#include <stdio.h>
union unionJob
{
    //defining a union
    char name[32];
    float salary;
    int workerNo;
} uJob;

struct structJob
{
    char name[32];
    float salary;
    int workerNo;
} sJob;

int main()
{
    printf("size of union = %d bytes", sizeof(uJob));
    printf("\nsize of structure = %d bytes", sizeof(sJob));
    return 0;
}

```

```
size of union = 32 bytes
size of structure = 40 bytes
--------------------------------
Process exited after 0.04807 seconds with return value 0
Press any key to continue . . .
```

DIFFERENCE BETWEEN STRUCTURE AND UNION IN C:

C Structure	C Union
Structure allocates storage space for all its members separately.	Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space
Structure occupies higher memory space.	Union occupies lower memory space over structure.
We can access all members of structure at a time.	We can access only one member of union at a time.

Structure example:
struct student
{
int mark;
char name[6];
double average;
};
	Union example:
	union student
	{
	int mark;
	char name[6];
	double average;
	};

For above structure, memory allocation will be like below. int mark – 2B //consider 2 Byte for int char name[6] – 6B double average – 8B Total memory allocation = 2+6+8 = 16 Bytes	For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types. Total memory allocation = 8 Bytes

SELF REFERENTIAL STRUCTURES

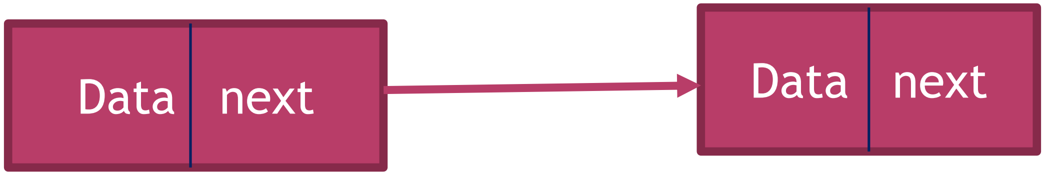
- A structure can have members which point to a structure variable of the same type.
- These types of structures are called self referential structures and are widely used in dynamic data structures like trees, linked list, etc.
- The following is a definition of a self referential structure.

```
struct node
{
        int data;
        struct node *next;
};
```

Here, next is a pointer to a struct node variable.

- It should be remembered that a pointer to a structure is similar to a pointer to any other variable.
- A self referential data structure is essentially a structure definition which includes at least one member that is a pointer to the structure of its own kind.
- struct node

```
{
        int data;
        struct node *next;
};
```



Data next

Data next

- Thank You