

Passing structure to function:

```
TC
Line 18 Col 9 Insert Indent Tab Fill Unindent * E:2PM.C
#include<stdio.h>
#include<conio.h>
struct stu
{
int id;
char name[20];
};
void show(struct stu s) /* fun def */
{
printf("id=%d, name=%s",s.id,s.name);
}
void main()
{
struct stu s={1,"Krish"};
clrscr();
show(s); /* fun calling */
getch();
}
```

Activate Windows
Go to PC settings to activate Windows.

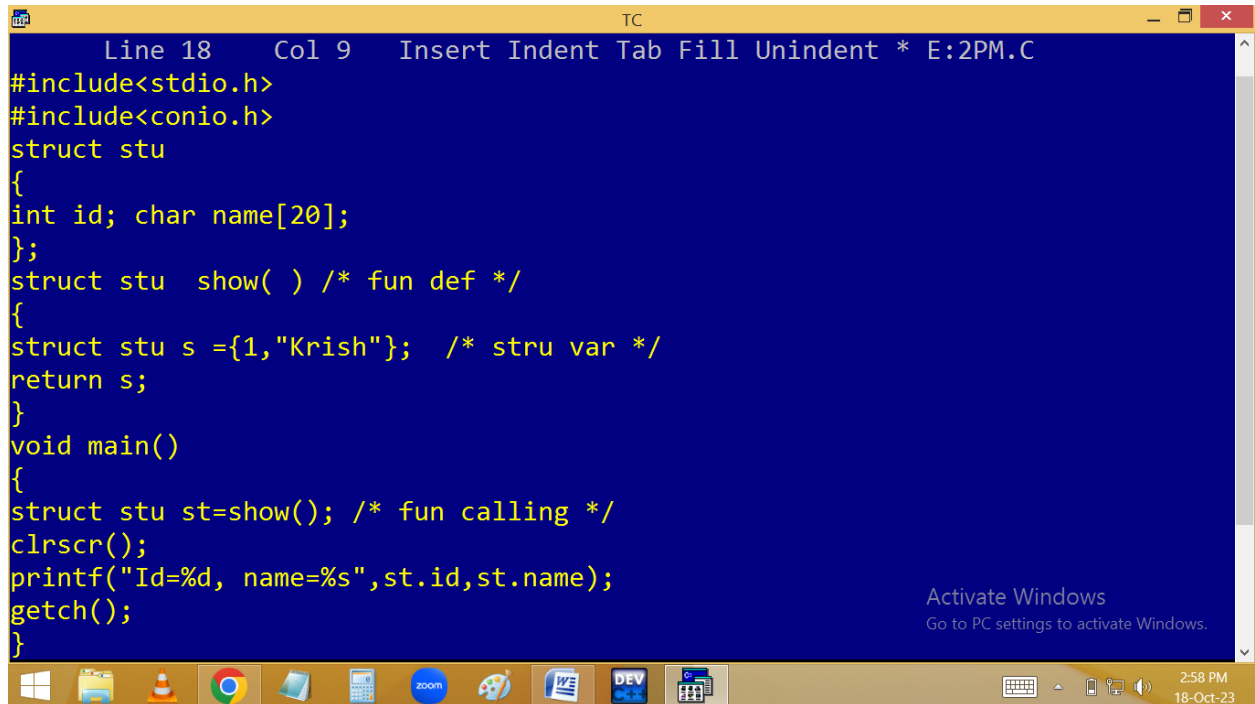
2:55 PM
18-Oct-23

```
TC
id=1, name=Krish_
```

Activate Windows
Go to PC settings to activate Windows.

2:55 PM
18-Oct-23

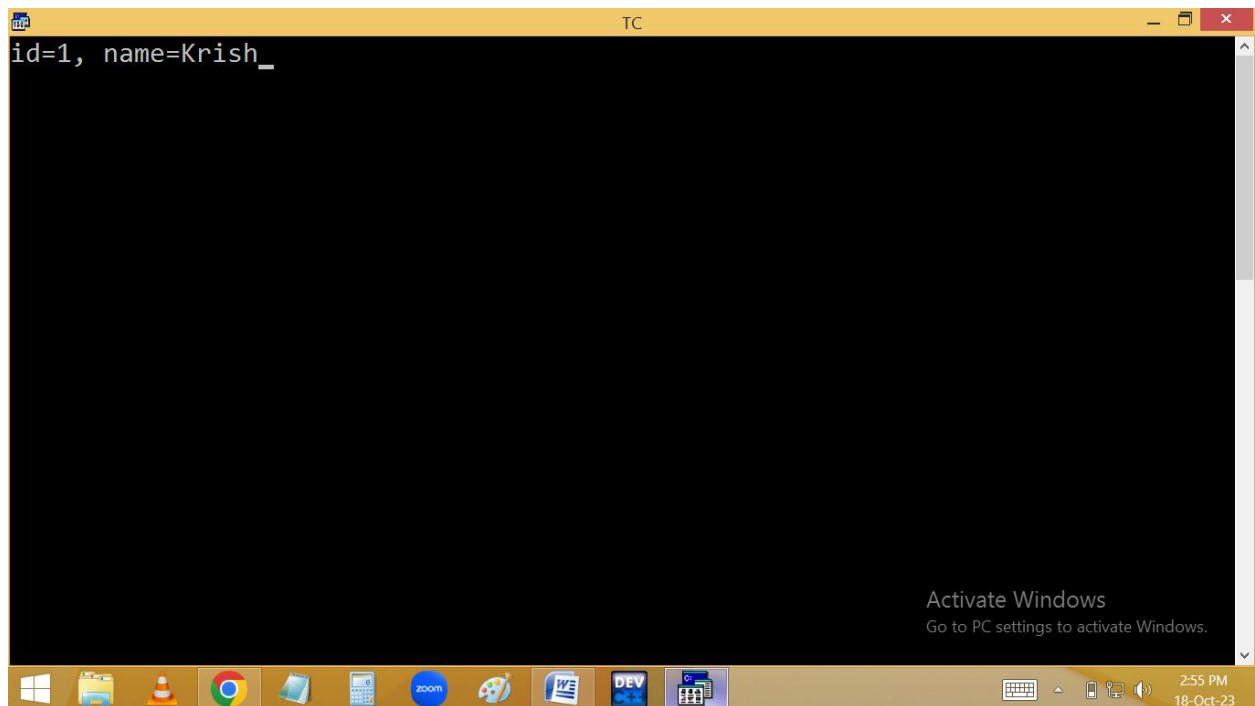
Function returning object:



```
Line 18   Col 9   Insert Indent Tab Fill Unindent * E:2PM.C
#include<stdio.h>
#include<conio.h>
struct stu
{
int id; char name[20];
};
struct stu show( ) /* fun def */
{
struct stu s ={1,"Krish"}; /* stru var */
return s;
}
void main()
{
struct stu st=show(); /* fun calling */
clrscr();
printf("Id=%d, name=%s",st.id,st.name);
getch();
}
```

Activate Windows
Go to PC settings to activate Windows.

2:58 PM
18-Oct-23



```
id=1, name=Krish_
```

Activate Windows
Go to PC settings to activate Windows.

2:55 PM
18-Oct-23

Nested / embedded structure:

Declaring a structure variable / structure within another structure. It allows the concept of reusability.

```
TC
#include<stdio.h>
#include<conio.h>
struct bank
{
int acno; char name[20];
};
struct loan
{
struct bank b; /* nested str var */
float amt, emi;
}l;
void main()
{
clrscr();
printf("Enter acno, name, loan amt, emi ");
scanf("%d %s %f %f", &l.b.acno,l.b.name,&l.amt,&l.emi);
printf("%s taken %.2f loan amount with emi of %.2f",l.b.name,l.amt,l.emi);
getch();
}_
```

Enter acno, name, loan amt, emi 101 Krish 10000000 50000
Krish taken 10000000.00 loan amount with emi of 50000.00

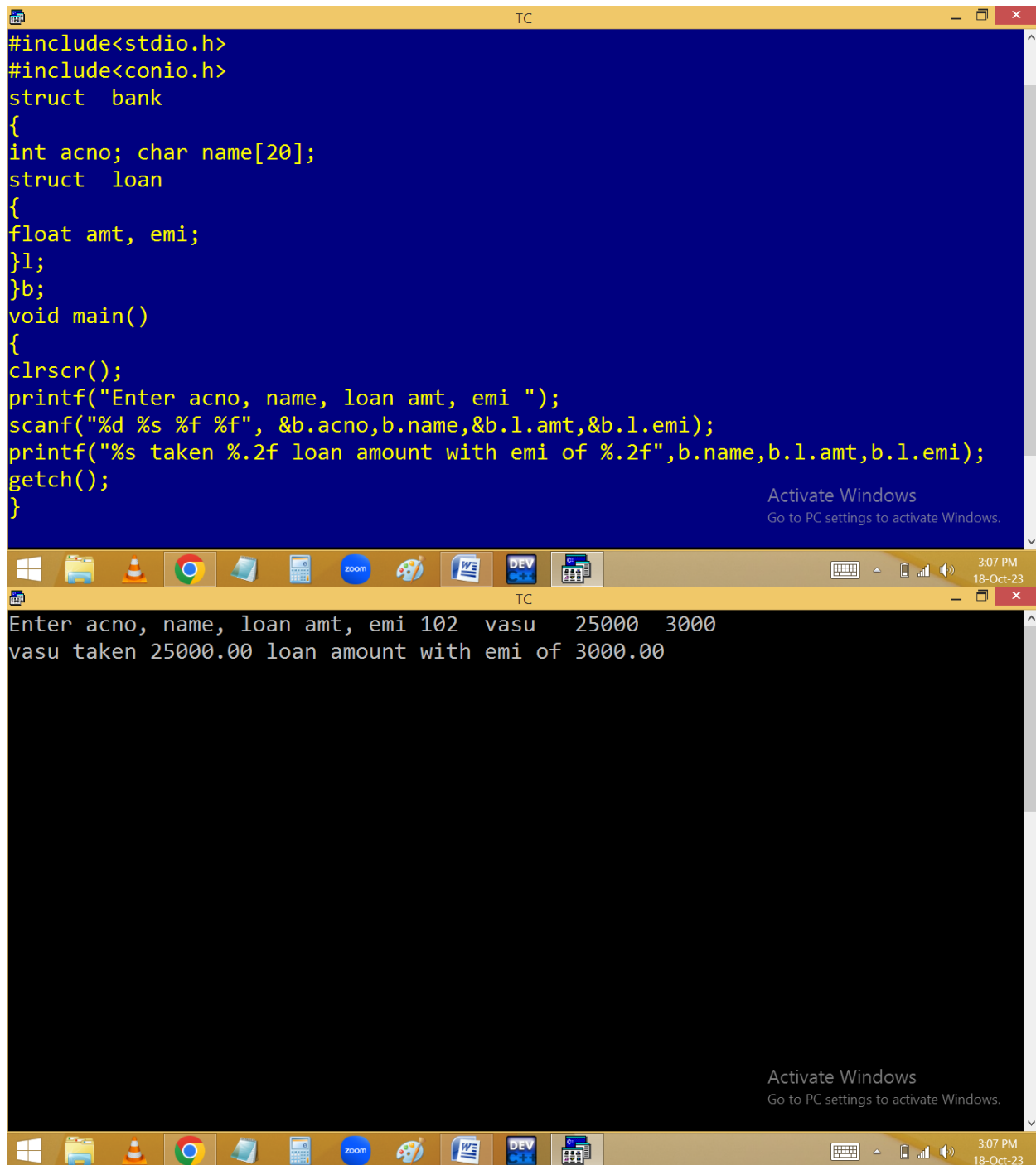
Activate Windows
Go to PC settings to activate Windows.

3:04 PM
18-Oct-23

TC

Activate Windows
Go to PC settings to activate Windows.

3:05 PM
18-Oct-23



```
#include<stdio.h>
#include<conio.h>
struct bank
{
int acno; char name[20];
struct loan
{
float amt, emi;
}l;
}b;
void main()
{
clrscr();
printf("Enter acno, name, loan amt, emi ");
scanf("%d %s %f %f", &b.acno,b.name,&b.l.amt,&b.l.emi);
printf("%s taken %.2f loan amount with emi of %.2f",b.name,b.l.amt,b.l.emi);
getch();
}
```

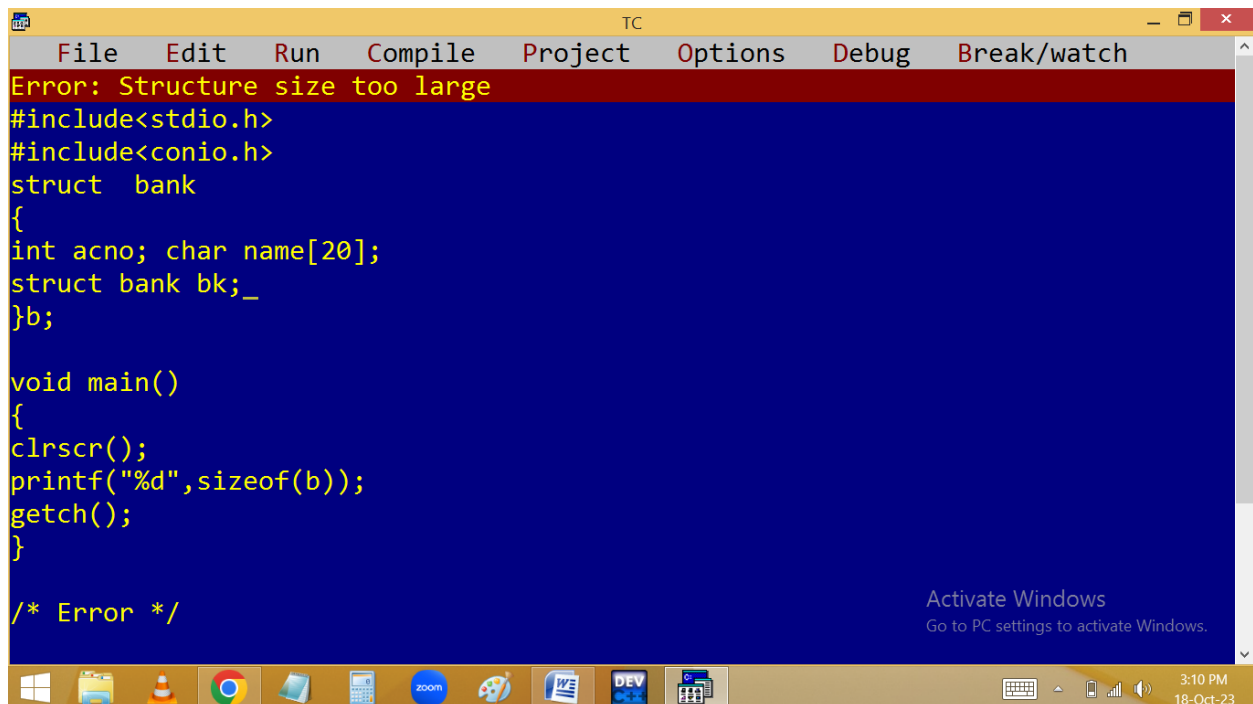
Enter acno, name, loan amt, emi 102 vasu 25000 3000
vasu taken 25000.00 loan amount with emi of 3000.00

Self referenced / self referential structure:

Declaring pointer structure variable of same structure within the structure is called self

referential structure. It is used in data structures to manage the linked list, stacks and queues.

It is not possible to create the normal structure variable within the same structure. Because of it causes memory overflow. In this situation we have to declare the structure variable using a pointer.



The screenshot shows a Turbo C++ (TC) IDE window. The title bar reads 'TC'. The menu bar includes 'File', 'Edit', 'Run', 'Compile', 'Project', 'Options', 'Debug', and 'Break/watch'. A red error message banner at the top states 'Error: Structure size too large'. The code editor has a dark blue background with yellow text. The code defines a structure named 'bank' containing an integer 'acno', a character array 'name' of size 20, and a pointer to the same structure 'struct bank bk;'. The 'main' function calls 'clrscr()', 'printf' to print the size of 'b', and 'getch()'. A comment '/* Error */' is at the bottom. The Windows taskbar at the bottom shows various icons including File Explorer, VLC, Chrome, and the system clock indicating 3:10 PM on 18-Oct-23.

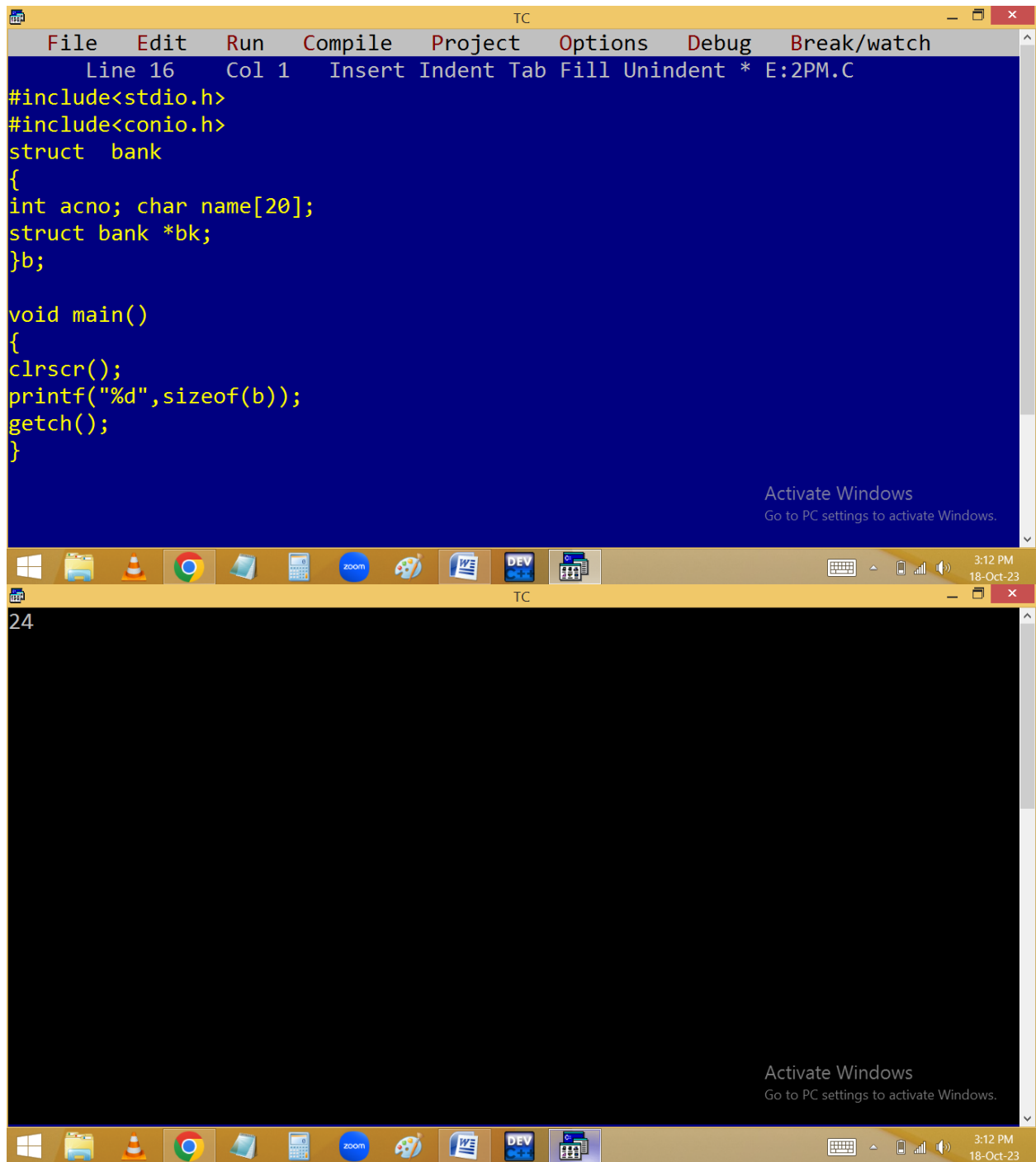
```
File Edit Run Compile Project Options Debug Break/watch
Error: Structure size too large
#include<stdio.h>
#include<conio.h>
struct bank
{
int acno; char name[20];
struct bank bk;_
}b;

void main()
{
clrscr();
printf("%d",sizeof(b));
getch();
}

/* Error */
```

Activate Windows
Go to PC settings to activate Windows.

3:10 PM
18-Oct-23



The image shows a screenshot of the Turbo C++ (TC) IDE. The main window displays a C program with the following code:

```
File Edit Run Compile Project Options Debug Break/watch
Line 16 Col 1 Insert Indent Tab Fill Unindent * E:2PM.C
#include<stdio.h>
#include<conio.h>
struct bank
{
int acno; char name[20];
struct bank *bk;
}b;

void main()
{
clrscr();
printf("%d",sizeof(b));
getch();
}
```

The status bar at the bottom of the IDE shows "Line 16 Col 1". The Windows taskbar is visible at the bottom, showing various application icons and the system clock indicating 3:12 PM on 18-Oct-23. An "Activate Windows" watermark is present in the bottom right corner of the IDE window.

Below the IDE window, there is a black console window with the number "24" displayed in the top left corner. The status bar at the bottom of the console window also shows "Activate Windows" and the system clock.

Coping structure data:

```
TC
File Edit Run Compile Project Options Debug Break/watch
Line 12 Col 32 Insert Indent Tab Fill Unindent * E:2PM.C
#include<stdio.h>
#include<conio.h>
struct bank
{
int acno; char name[20];
}b1={1,"Krish"},b2;

void main()
{
b2=b1;
clrscr();
printf("id=%d, name=%s",b2.acno, b2.name);
getch();
}
```

Activate Windows
Go to PC settings to activate Windows.

3:16 PM
18-Oct-23

```
TC
id=1, name=Krish_

```

Activate Windows
Go to PC settings to activate Windows.

3:16 PM
18-Oct-23

UNION

It is a user defined data type.

union is a keyword.

Like the structures unions are also used to place several variables of different data types under one name. But the main difference is all union members are having common memory/ one memory.

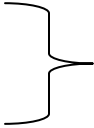
In structure memory allocated for all the structure members separately. But in union one memory allocated and it is used by all the union members. We can access all the structure members at a time. But in union one member is active at a time. Structure size is sum of all the structure members. Union size is biggest union member size.

There is a situation where we have several options and user has to select one then go for union.

Syntax:

```
union [ <union-tag-name> ]  
{
```

```
datatype  variable1;  
datatype  variable2;  
} union variables;
```

 **Union members**

➔ Declaration and Accessing of union members is similar to the structures.

Eg: Finding union size:

```
#include<stdio.h>  
#include<conio.h>  
  
union  stu  
{  
int id;  
char name[20];  
float fee;  
}  
  
void main()  
{  
printf("Union size  %d bytes", sizeof(s));
```

```
getch();  
}
```

Output: Union size 20 bytes.

Eg:

Direct Initialization of union members:

Note: In union only the first member is initialized directly.

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

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

```
union stu
```

```
{
```

```
int id;
```

```
char name[20];
```

```
float fee;
```

```
}s={1001,"krish",10000};
```

```
void main()
```

```
{  
clrscr();  
printf("Id=%d\nName=%s\nFee=%.2f",s.id,  
s.name, s.fee);  
getch();  
}
```

Output: Initializer syntax error

Solution: s = {1001} ;

Eg:

Accessing union members:

```
#include<stdio.h>  
#include<conio.h>  
union sample  
{  
int a;  
long int b;
```

```
};  
void main()  
{  
union sample s;  
clrscr();  
s.a=100;  
printf("a = %d\n", s.a);  
s.b=200;  
printf("b = %ld", s.b);  
getch();  
}
```

Output

a = 100

b = 200

Differences between structure and union:

STRUCTURE	UNION
All the structure	Only one union member

members are active at a time.	is active at a time.
memory is allocated for all the structure members	memory allocated for the variable, which requires more memory in the union.
All the structure members initialized at a time.	Only the first member of union is initialized.
Structure size is sum of all the structure members.	Union size is the biggest variable data type size in the entire union.
They are useful to declare a compound data type to group data members related to person or item etc.	It is useful in certain situation where the user will select any one data member from group of data members.

enum / enumeration

It allows to store several integer values in the form of identifiers.

It is a user defined data type.

enum is a keyword.

It is similar to the structure, which allows to store several values. But the difference is it stores only the integers in text format [identifiers].

Syntax:

```
enum [<tag_name>]
{
Identifier1, Identifier2, ....
}
[Variable=value];
```

Here the identifier numbers started with 0 ,1, 2,..and end with N-1 implicitly. We can change this series manually.

Eg: 1

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

```
enum week
{
sun, mon, tue, wed, thu, fri, sat
}
day=sun;
void main()
{
clrscr();
printf("Sunday is %dst day of the
week",day+1);
getch();
}
```

Output: Sunday is 1st day of the week

Eg: 2

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



```
enum
{
    police=100, fire, ambulance=108
}e=fire;

void main()
{
    clrscr();
    printf("Dialing %d service...", e);
    getch();
}
```

Output: dialing 101 service...

Eg: 3

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

enum colors {black, blue, green, cyan, red};

void main()
```

```
{  
clrscr();  
enum colors color=green;  
textcolor(color);  
textbackground(1);  
cprintf("Naresh It");  
getch();  
}
```