6.0 Self-referential structure

A self-referential is one which contains a pointer to its own type. This type of structure is also known as linked list. For example

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

Defines a data type, struct node. A structure type struct node has two members-integer member data and pointer member nextptr. Member nextptr points to a structure of type struct node – a structure of the same type as one being declared here, hence the term “self-referential structure”. Member nextptr referred to as a link i.e. nextptr can be used to tie a structure of type struct node to another structure of the same type. self referential structures can be linked together to form a usual data structures such as lists etc.

```
#include<stdio.h>
struct student
{
        char name[30];
        int age;
        char address[20];
        struct student *next;
};
int main()
{
    struct student st1={"lakki",24,"Vizag"};
    struct student st2={"lakki1",21,"Nuzvid "};
    struct student st3={"praveen",22,"Hyd"};
    st1.next = &st2;
    st2.next = &st3;
    st3.next = NULL;
    printf("Student Name:%s\tAge:%d\tAddress:%s\n",st1.name,st1.age,st1.address);
    printf("Student 1 stored at %x \n",st1.next);
    printf("Student Name:%s\tAge:%d\tAddress:%s\n",st2.name,st2.age,st2.address);
    printf("Student 2 stored at %x \n",st2.next);
    printf("Student Name:%s\tAge:%d\tAddress:%s\n",st3.name,st3.age,st3.address);
    printf("Student 3 stored at %x \n",st3.next); return 0;
}
```

Output

```
Student Name:lakki          Age:24          Address: Vizag
Student 1 stored at 88f0
Student Name:lakki1         Age:21          Address:Nuzvid
Student 2 stored at 8854
Student Name:Praveen        Age:22          Address: Hyd
Student 3 stored at 0
```

To Create Structure using malloc( ) :

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
	int n,i;
	printf("Enter no of students=");
	scanf("%d",&n);
	struct student
	{
		int regid;
		char name[100];
		float cgpa;
	};
	struct student*ptr;
	ptr=(struct student*)malloc(n*sizeof(struct student));
	if(ptr==NULL)
	{
		printf("no storage left on your device");
		exit(1);
	}
	else
	{
		for(i=0;i<n;i++)
		{
			printf("enter REGID=");
			scanf("%d",&(ptr+i)->regid);
			printf("enter name=");
			scanf("%s",&(ptr+i)->name);
			printf("enter cgpa=");
			scanf("%f",&(ptr+i)->cgpa);
		}
		Printf("\nStudent details=");
		for(i=0;i<n;i++)
		{
		printf("regid=%d\n",(ptr+i)->regid);
		printf("name=%s\n",(ptr+i)->name);
		printf("cgpa=%.1f\n\n",(ptr+i)->cgpa);
		}
	}
}
```

Output:-

```
Enter no of students=2
enter  REGID=123
enter name=Ram
enter cgpa=9.8
enter REGID=456
enter name=abc
enter cgpa=9.5

Student details=
REGID=123
name=Ram
cgpa=9.8

 REGID=456
name=abc
cgpa=9.5
```

Note:-

If pointer variable is pointing to the structure we are using ->operator to read the structure members.

Unions

A *Union* is a user defined data type like structure. The union groups logically related variables into a single unit.

- ✓ In structure each member has its own memory location where as the members of union has the same memory location.
- ✓ We can assign values to only one member at a time, so assigning value to another member that time has no meaning.

When a union is declared, compiler allocates memory locations to hold largest data type member in the union. So, union is used to save memory. Union is useful when it is not necessary to assign the values to all the members of the union at a time.

Union can be declared as

```
union
{
        member 1;
        member 2;
        … …
        … …
        member N;
};
```

Following example shows how to declare union and accessing the members of union

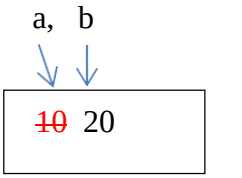
```
#include<stdio.h>
union student
{
        char name[30];
        int age;
};
int main()
{
        union student st;
        clrscr();
        printf("Enter student name:");
        scanf("%s",st.name);
        printf("Student Name:%s, age=
        %d\n",st.name,st.age); st.age=24;
        printf("Student Name:%s, age=
        %d\n",st.name,st.age);
        getch();
        return 0;
}
```

Output

Enter student name: Ram
Student Name: Ram, age= 26966
Student Name: ¶ , age= 24

Example 2:

```
#include<stdio.h>
Union un
{
int a,b;
}var;
int main( )
{
var.a=10;
pf("A=%d \t B=%d",var.a,var.b);
var.b=20;
pf("A=%d \t B=%d",var.a,var.b);
}
```



2046 (2 bytes)

OUTPUT:

A=10 B=10

A=20 B=20

Look at the output

First time union member *name* has a value which is inputted through the keyboard. So *name* is displayed. Next time we were assigned a values to age, union will lost the value in member *name*, so this time only student‟s age is displayed.

Following figure illustrates how members of structure and union are stored.

```
struct student
{
        char name[15];
        int age;
};
```

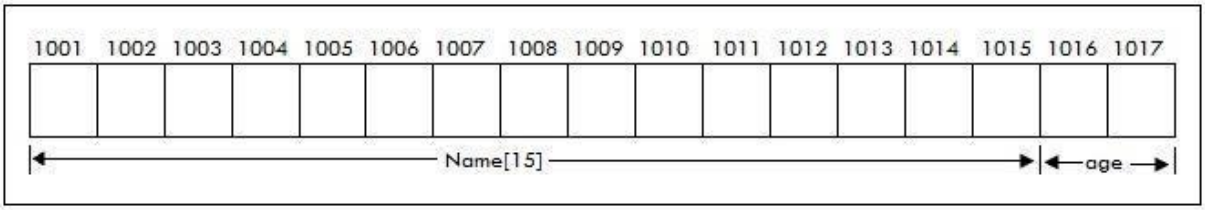


Fig. Storage in structure

```
union student
{
        char name[15];
        int age;
};
```

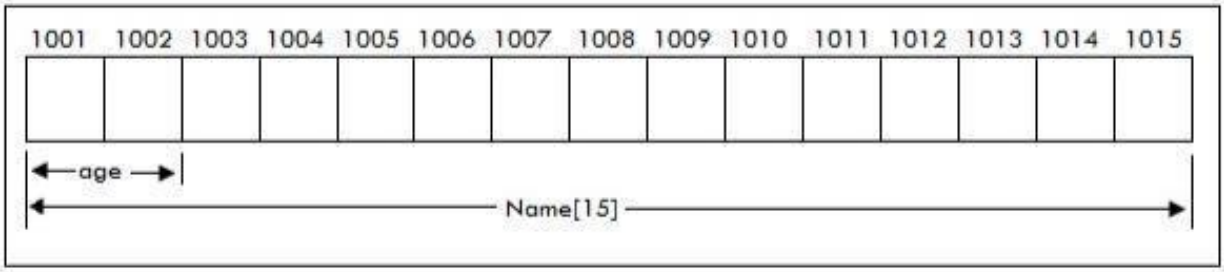


Fig. Storage in union

Comparison between structure and union

S.No	Structure	Union
1.	Stores heterogeneous data	Stores heterogeneous data
2.	Members are stored separately in memory locations.	Members are stored in same memory location.
3.	In structures all the members are available.	In union only one member is available at a time.
4.	Occupies more memory when compared to union	Occupies less memory when compared with structure.
5.	Size of the structure is the total memory space required by its members	Size of the union is the size of the largest data type member in the union. All the elements share same memory

		location.All the elements we cannot process.

	Ex:
	struct student
	{
	int sid;
	char sname[10];
	float smarks;
	};
	
	Struct student st;
	
	sid sname smarks
	[ ] [ ] [ ]
	2bytes 10 bytes 4 bytes
		Ex:
		unioin student
		{
		int sid;
		char sname[10];
		float smarks;
		};
		
		union student st;
		
		sid ,sname , smarks
		[ ]
		10 bytes

Comparison between array and structure

S.No	Array	Structure
1.	Stores homogeneous data	Stores heterogeneous data
2.	Two arrays of same type cannot be assigned to one to one	Structures of same type can be assigned.
3.	Arrays can be initialized	Structures cannot be initialized
4.	Array is a combination of elements	Structure is a combination of members
5.	Multidimensional arrays are possible	No in case of structures
6.	No operators are required to access elements of an array	Operators like „.“ or „->“ are required to access members of a structure.
7.	An element in the array referenced by specifying array name and its position in the array. For example: a[5]	Members in a structure will be referenced as structurename.membername
8.	C treats array names as pointers	Structure name is not treated as pointer variable.
9.	When an array name is passed as argument to function, it is call by reference.	When an structure name is passed as argument to function, it is call by value.

Array of Unions:

```
#include<stdio.h>
union num
{
        int a;
        int b;
};
int main()
{
        int i;
        union num arr[3];
        arr[0].a=10;
        arr[0].b=20;
        arr[1].a=30;
        arr[1].b=40;
        arr[2].a=50;
        arr[2].b=60;
        for(i=0;i<3;i++)
        {
        printf("\n the a and b values in index %d is %d and %d ",i,arr[i].a,arr[i].b);

        }
        return 0;
}
```

OUTPUT:
the a and b values in index 0 is 20 and 20
the a and b values in index 1 is 40 and 40
the a and b values in index 0 is 60 and 60

Union inside the structures:

```
#include<stdio.h>
struct student
{
        union st
        {
          char name[20];
          int sid;
   }st1;
   int marks;

};
int main()
{
        struct student st;
        char choice;
        printf("\n you can enter the NAME or ID:");
```

```
            printf("\n Do you want Ur results by Name Enter (Y) :");
            scanf("%c",&choice);
            if(choice=='N'||choice=='Y')
            {
            printf("\n enter Name:");
            scanf("%s",st.st1.name); // or st.name
            printf("\n enter marks:");
            scanf("%d",&st.marks);
            printf("\ Student Name:%s",st.st1.name);
            printf("\nStudent matrks:%d",st.marks);
  }

else
{
            printf("\n enter sid:");
            scanf("%d",&st.st1.sid);
            printf("\n enter marks:");
            scanf("%d",&st.marks);
            printf("\ Student Id:%d",st.st1.sid);
            printf("\nStudent matrks:%d",st.marks);
}
}
```

OUTPUT:

```
you can enter the NAME or ID:
Do you want Ur results by Name Enter (Y): Y
Enter name: Ram
Enter marks: 24
Student Name: Ram
Student Marks: 24

you can enter the NAME or ID:
Do you want Ur results by Name Enter (Y): N
Enter ID: 1224
Enter marks: 78
Student ID:1224
Student Marks: 78
```

Note: When we are declare the union inside the structure , no need to declare variable for union, by accessing union members it also possible to access through structure variable.

St.un.member **or** st.member

Structure.union.member

Structures inside the Unions:

```
 #include<stdio.h>
struct a
{
        int marks;
        int sid;
};
struct b
{
   char name[20];
        int marks;

};
union Student
{
        struct a A;
        struct b B;

};
int main()
{
        union Student s;
        char ch;
        printf("\n you can enter the NAME or ID:");
        printf("\n Do you want Ur results by Enter Name or Roll No (N/R) :");
        scanf("%c",&ch);
        if(ch=='N'||ch=='n')
        {
        printf("\n enter Name:");
        scanf("%s",s.B.name);
        printf("\n enter marks:");
        scanf("%d",&s.B.marks);
        printf("\ Student Name:%s",s.B.name);
        printf("\nStudent matrks:%d",s.B.marks);
  }
  if(ch=='R'||ch=='r')
        {
        printf("\n enter RoolNo:");
        scanf("%d",&s.A.sid);
        printf("\n enter marks:");
        scanf("%d",&s.A.marks);
        printf("\ Student ID:%d",s.A.sid);
        printf("\nStudent matrks:%d",s.A.marks);
  }
return 0;
```

}

OUTPUT:

```
you can enter the NAME or ID:
Do you want Ur results by Enter Name or Roll No: (N/R) : N
Enter name: Ram
Enter marks: 24
Student Name: Ram
Student Marks: 24

you can enter the NAME or ID:
Do you want Ur results by Enter Name or Roll No: (N/R) : N
Enter ID: 1224
Enter marks: 78
Student ID:1224
Student Marks: 78
```