Output:-i=0,2,4,3,2.

When there is large program i.e divided into several files, then external variable should be preferred. External variable extend the scope of variable.

Lecture Note: 20

POINTER

A pointer is a variable that store memory address or that contains address of another variable where addresses are the location number always contains whole number. So, pointer contain always the whole number. It is called pointer because it points to a particular location in memory by storing address of that location.

Syntax-

Data type *pointer name;

Here * before pointer indicate the compiler that variable declared as a pointer.

e.g.

```
int *p1; //pointer to integer type
float *p2; //pointer to float type
char *p3; //pointer to character type
```

When pointer declared, it contains garbage value i.e. it may point any value in the memory.

Two operators are used in the pointer i.e. **address operator(&)** and **indirection operator or dereference operator (*).**

Indirection operator gives the values stored at a particular address.

Address operator cannot be used in any constant or any expression.

Example:

```
    void main()
    {
     int i=105;
     int *p;
     p=&i;
t
 printf("value of i=%d",*p);
 printf("value of i=%d",*/&i);
 printf("address of i=%d",&i);
 printf("address of i=%d",p);
 printf("address of p=%u",&p);
}
```

Pointer Expression

Pointer assignment

int i=10;

int *p=&i;//value assigning to the pointer

Here declaration tells the compiler that P will be used to store the address of integer value or in other word P is a pointer to an integer and *p reads the **value at the address contain in p.**

P++;

printf("value of p=%d");

We can assign value of 1 pointer variable to other when their base type and data type is same or both the pointer points to the same variable as in the array.

Int *p1,*p2;

P1=&a[1];

P2=&a[3];

We can assign constant 0 to a pointer of any type for that symbolic constant '**NULL**' is used such as

*p=NULL;

It means pointer doesn't point to any valid memory location.

Pointer Arithmetic

Pointer arithmetic is different from ordinary arithmetic and it is perform relative to the data type(base type of a pointer).

Example:-

If integer pointer contain address of 2000 on incrementing we get address of 2002 instead of 2001, because, size of the integer is of 2 bytes.

Note:-

When we move a pointer, somewhere else in memory by incrementing or decrement or adding or subtracting integer, it is not necessary that, pointer still pointer to a variable of same data, because, memory allocation to the variable are done by the compiler.

But in case of array it is possible, since there data are stored in a consecutive manner.

Ex:-

```
void main( )
{
static int a[ ]={20,30,105,82,97,72,66,102};
int *p,*p1;
P=&a[1];
P1=&a[6];
printf("%d",*p1-*p);
printf("%d",p1-p);
}
```

Arithmetic operation never perform on pointer are:

addition, multiplication and division of two pointer.

multiplication between the pointer by any number.

division of pointer by any number

-add of float or double value to the pointer.

Operation performed in pointer are:-

/* Addition of a number through pointer */

Example

```
int i=100;
int *p;
```

p=&i;

p=p+2;

p=p+3;

p=p+9;

ii /* Subtraction of a number from a pointer'*/

Ex:-

int i=22;

*p1=&a;

p1=p1-10;

p1=p1-2;

iii- Subtraction of one pointer to another is possible when pointer variable point to an element of same type such as an array.

Ex:-

in tar[ ]={2,3,4,5,6,7};

int *ptr1,*ptr1;

ptr1=&a[3]; //2000+4

ptr2=&a[6]; //2000+6

Lecture Note: 21

Precedence of dereference (*) Operator and increment operator and decrement operator

The precedence level of difference operator increment or decrement operator is same and their associatively from right to left.

Example :-

int x=25;

int *p=&x;

Let us calculate int y=*p++;

Equivalent to *(p++)

Since the operator associate from right to left, increment operator will applied to the pointer p.

i) int y=*p++; equivalent to *(p++)

p =p++ or p=p+1

ii) *++p;→*(++p)→p=p+1

y=*p

iii) int y=++*p

equivalent to ++(*p)

p=p+1 then *p

iv) y=(*p)++→equivalent to *p++

y=*p then

P=p+1 ;

Since it is postfix increment the value of p.

Pointer Comparison

Pointer variable can be compared when both variable, object of same data type and it is useful when both pointers variable points to element of same array.

Moreover pointer variable are compared with zero which is usually expressed as null, so several operators are used for comparison like the relational operator.

==,!=,<=,<,>,>=, can be used with pointer. Equal and not equal operators used to compare two pointer should finding whether they contain same address or not and they will equal only if are null or contains address of same variable.

Ex:-

```
  void main()
{
static int arr[]={20,25,15,27,105,96}
int *x,*y;
x=&a[5];
y=&(a+5);
if(x==y)
printf("same");
else
printf("not");


}
```

Lecture Note: 22

Pointer to pointer

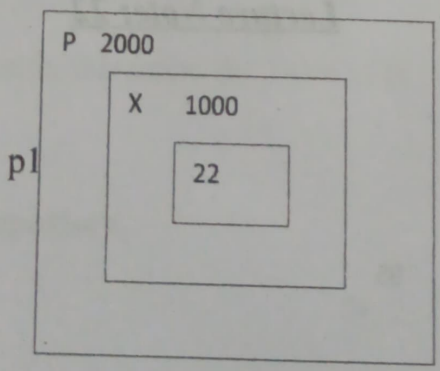
Addition of pointer variable stored in some other variable is called pointer to pointer variable.

Or

Pointer within another pointer is called pointer to pointer.

Syntax:-

```
Data type **p;
int x=22;
int *p=&x;
int **p1=&p;
printf("value of x=%d",x);
printf("value of x=%d",*p);
printf("value of x=%d",*&x);
printf("value of x=%d",**p1);
printf("value of p=%u",&p);
printf("address of p=%u",p1);
printf("address of x=%u",p);
printf("address of p1=%u",&p1);
printf("value of p=%u",p);
printf("value of p=%u",&x);
```



3000

Pointer vs array

Example :-

```
  void main()
{
static char arr[]="Rama";
char*p="Rama";
printf("%s%s", arr, p);
```

In the above example, at the first time printf( ), print the same value array and pointer.

Here array arr, as **pointer to character** and **p act as a pointer to array of character .** When we are trying to increase the value of arr it would give the error because its known to compiler about an array and its base address which is always printed to base address is known as constant pointer and the base address of array which is not allowed by the compiler.

```
printf("size of (p)",size of (ar));
```

size of (p) 2/4 bytes

size of(ar) 5 byes

Sructure

It is the collection of dissimilar data types or heterogenous data types grouped together. It means the data types may or may not be of same type.

Structure declaration-

```
struct tagname
{
Data type member1;
Data type member2;
Data type member3;
.....
.....
Data type member n;
};
```

OR

```
struct
{
Data type member1;
Data type member2;
```

```
Data type member3;
.....
.....
Data type member n;
};
```

OR

```
struct tagname
{
struct element 1;
struct element 2;
struct element 3;
.....
.....
struct element n;
};
```

Structure variable declaration;

```
struct student
{
        int age;
char name[20];
char branch[20];
```

}; struct student s;

Initialization of structure variable-

Like primary variables structure variables can also be initialized when they are declared. Structure templates can be defined locally or globally. If it is local it can be used within that function. If it is global it can be used by all other functions of the program.

We cant initialize structure members while defining the structure

```
struct student
{
    int age=20;
char name[20]="sona";
}s1;
```

The above is **invalid.**

A structure can be initialized as

```
struct student
{
    int age,roll;
char name[20];
} struct student s1={16,101,"sona"};
  struct student s2={17,102,"rupa"};
```

If initialiser is less than no.of structure variable, automatically rest values are taken as zero.

Accessing structure elements-

Dot operator is used to access the structure elements. Its associativety is from left to right.

structure variable ;

s1.name[];

s1.roll;

s1.age;

Elements of structure are stored in contiguous memory locations. Value of structure variable can be assigned to another structure variable of same type using assignment operator.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int roll, age;
char branch;
} s1,s2;
printf("\n enter roll, age, branch=");
scanf("%d %d %c", &s1.roll, &s1.age, &s1.branch);
s2.roll=s1.roll;
printf(" students details=\n");
printf("%d %d %c", s1.roll, s1.age, s1.branch);
printf("%d", s2.roll);
```

}

Unary, relational, arithmetic, bitwise operators are not allowed within structure variables.

Lecture Note:24

Size of structure-

Size of structure can be found out using sizeof() operator with structure variable name or tag name with keyword.

sizeof(struct student); or

sizeof(s1);

sizeof(s2);

Size of structure is different in different machines. So size of whole structure may not be equal to sum of size of its members.

Array of structures

When database of any element is used in huge amount, we prefer Array of structures.

Example: suppose we want to maintain data base of 200 students, Array of structures is used.

#include<stdio.h>

#include<string.h>

struct student

{

```
char name[30];
char branch[25];
int roll;
};
void main()
{
struct student s[200];
int i;
s[i].roll=i+1;
printf("\nEnter information of students:");
for(i=0;i<200;i++)
{
printf("\nEnter the roll no:%d\n",s[i].roll);
printf("\nEnter the name:");
scanf("%s",s[i].name);
printf("\nEnter the branch:");
scanf("%s",s[i].branch);
printf("\n");
}
printf("\nDisplaying information of students:\n\n");
for(i=0;i<200;i++)
{
printf("\n\nInformation for roll no%d:\n",i+1);
```

```
printf("\nName:");
puts(s[i].name);
printf("\nBranch:");
puts(s[i].branch);
}
}
```

In Array of structures each element of array is of structure type as in above example.

Array within structures

```
struct student
{
char name[30];
int roll,age,marks[5];
}; struct student s[200];
```

We can also initialize using same syntax as in array.

Nested structure

When a structure is within another structure, it is called Nested structure. A structure variable can be a member of another structure and it is represented as

struct student

```
{
element 1;
element 2;
.....
.....
struct student1
{
member 1;
member 2;
}variable 1;
.....
.....
element n;
}variable 2;
```

It is possible to define structure outside & declare its variable inside other structure.

```
struct date
{
int date,month;
};
struct student
{
```

```
 char nm[20];
int roll;
struct date d;
}; struct student s1;
   struct student s2,s3;
```

Nested structure may also be initialized at the time of declaration like in above example.

```
struct student s={"name",200, {date, month}};
                 {"ram",201, {12,11}};
```

Nesting of structure within itself is not valid. Nesting of structure can be extended to any level.

```
struct time
{
int hr,min;
};
struct day
{
int date,month;
struct time t1;
};
struct student
```

```
{
char nm[20];
struct day d;
}stud1, stud2, stud3;
```

Lecture Note: 25

Passing structure elements to function

We can pass each element of the structure through function but passing individual element is difficult when number of structure element increases. To overcome this, we use to pass the whole structure through function instead of passing individual element.

```
#include<stdio.h>
#include<string.h>
void main()
{
struct student
{
char name[30];
char branch[25];
int roll;
}struct student s;
printf("\n enter name=");
```

```
gets(s.name);
printf("\nEnter roll:");
scanf("%d",&s.roll);
printf("\nEnter branch:");
gets(s.branch);
display(name,roll,branch);
}
display(char name, int roll, char branch)
{
printf("\n name=%s,\n roll=%d, \n branch=%s", s.name, s.roll. s.branch);
}
```

Passing entire structure to function

```
#include<stdio.h>
#include<string.h>
struct student
{
char name[30];
int age,roll;
};
display(struct student);                    //passing entire structure
void main()
```

```
{
  struct student s1={"sona",16,101 };
  struct student s2={"rupa",17,102 };
display(s1);
display(s2);
}
display(struct student s)
{
printf("\n name=%s, \n age=%d ,\n roll=%d", s.name, s.age, s.roll);
}
```

Output: name=sona
roll=16

Lecture Note: 26

UNION

Union is derived data type contains collection of different data type or dissimilar elements. All definition declaration of union variable and accessing member is similar to structure, but instead of keyword struct the keyword union is used, the main difference between union and structure is

Each member of structure occupy the memory location, but in the unions members share memory. Union is used for saving memory and concept is useful when it is not necessary to use all members of union at a time.

Where union offers a memory treated as variable of one type on one occasion where (struct), it read number of different variables stored at different place of memory.

Syntax of union:

union student

{

datatype member1;

datatype member2;

};

Like structure variable, union variable can be declared with definition or separately such as

union union name

{

Datatype member1;

}var1;

Example:- union student s;

Union members can also be accessed by the dot operator with union variable and if we have pointer to union then member can be accessed by using (arrow) operator as with structure.

Example:- struct student

struct student

{

int i;

char ch[10];

};struct student s;

Here datatype/member structure occupy 12 byte of location is memory, where as in the union side it occupy only 10 byte.

Lecture Note:27

Nested of Union

When one union is inside the another union it is called nested of union.

Example:-

union a

{

int i;

int age;

};

union b

```
{
char name[10];
union a aa;
}; union b bb;
```

There can also be union inside structure or structure in union.

Example:-

```
 void  main()
  {
  struct a
 {
int i;
char ch[20];
};
struct  b
{
int i;
char d[10];
};
union z
{
struct a a1;
struct b b1;
```

```
}; union z z1;
z1.b1.j=20;
z1.a1.i=10;
z1.a1.ch[10]= " i";
z1.b1.d[0]="j ";
printf(" ");
```

Dynamic memory Allocation

The process of allocating memory at the time of execution or at the runtime, is called dynamic memory location.

Two types of problem may occur in static memory allocation.

If number of values to be stored is less than the size of memory, there would be wastage of memory.

If we would want to store more values by increase in size during the execution on assigned size then it fails.

Allocation and release of memory space can be done with the help of some library function called dynamic memory allocation function. These library function are called as **dynamic memory allocation function.** These library function prototype are found in the header file, "alloc.h" where it has defined.

Function take memory from memory area is called heap and release when not required.

Pointer has important role in the dynamic memory allocation to allocate memory.

malloc():