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);

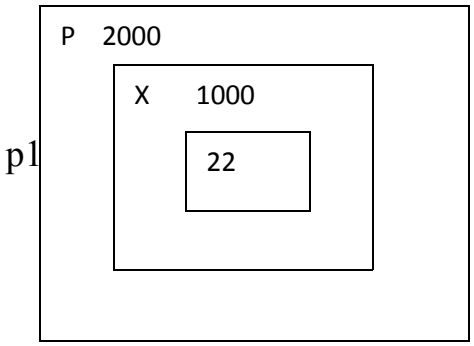
printf("address of p=%u",p1);

printf("address of x=%u",p);

printf("address of p1=%u",&p1);

printf("value of p=%u",p);

printf("value of p=%u",&x);



*Under revision

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