**Data Structure theory assignment 1**

**PRN:-2019BTEIT00023 Name:-Sumit Sunil Koundanya Class:-SYIT**

**Q.1)** What is meant by pointer and why it can be used?

Pointer is the variable which stores the address of another variable.

For ex- int \*ptr, char \*pt etc. pointers are used for following purposes:-

1. To return more than one value from a function.
2. To pass arrays and strings more conveniently from one function to another.
3. To allocate memory dynamically and access it.
4. To manipulate array by moving pointers to them instead of moving the arrays themselves.
5. To create complex data structures such as linked list, where one data structure contains reference to other data structure.

**Q.2)** How to declare and access a pointer variable?

Declaration of pointer variable

* In -directional Operator To access the value stored in the address we use the unary operator (\*) that returns the value of variable located at the address specified by its operand .this is also known as Dereferencing.

Syntax:-

Data type \*pointer\_ name;

Example:-

int \*ip; // pointer to an integer

double \*dp; // pointer to a double

float \*fp; // pointer to a float

char \*ch // pointer to a character

* Assignment Operator/Ampersand

To access address of a variable to a pointer, we use the unary operator (&) that returns the address of that variable. for example &x gives us address of variable x.

Syntax:-

Pointer variable\_name= & variable \_name;

For example- a b

int \*a;

1000

1

50

int b=&a;

1000 2000

Accessing:-

Program to use of pointer

#include<stdio.h>

void main()

{ x ptr

int x=10;

1210

10

int \*ptr;

ptr=&x; 1210 1500

printf(“\n%u”,ptr); //Address of x

printf(“\n%d”,\*ptr); // value at ptr

return ;

}

**Q.3)** Explain pointer increment/Decrement

1) Each time a pointer is incremented, it pointed to the memory location of next element of its data type. Increment operator when used with a pointer variable returns next address pointed by the pointer. The next address returned is the sum of current pointed address and size of pointer data type.

2) Each time a pointer is decremented, it points to the location of previous element. Decrement operator returns the previous address pointed by the pointer. The returned address is the difference of current pointed address and size of pointer data type.

3) All other pointers will increase or decrease depending on length of data type they are pointing to. If integer takes 2 bytes , increment to an integer pointer will be by 2 bytes and for float it is by 4 bytes.

int num=5; // Suppose address of num = 0x1230

int \*ptr; // Pointer variable

ptr = &num; // ptr points to 0x1230 or ptr points to num

ptr++; // ptr now points to 0x1234, since integer size is 4 bytes

ptr--; // ptr now points to 0x1230

**Q.4**) Explainthe use of pointers with Array

The address of Array element can be expressed in two ways:-

* By writing the actual array element preceded by the ampersand (&) sign.
* By writing an expression in which the subscript is added to the array name.

#include<stdio.h>

int main()

{

int \*p;

int (\*ptr)[5];

int arr[5];

p=arr;

ptr=&arr;

printf("p = %p, ptr = %p\n", p, ptr);

p++;

ptr++;

       printf("p = %p, ptr = %p\n", p, ptr);

return 0;

}

Output:-

p = 0x7fff4f32fd50, ptr = 0x7fff4f32fd50

p = 0x7fff4f32fd54, ptr = 0x7fff4f32fd64

***p***: is pointer to 0th element of the array *arr*, while ***ptr*** is a pointer that points to the whole array *arr*.

* The base type of *p* is int while base type of *ptr* is ‘an array of 5 integers’.
* We know that the pointer arithmetic is performed relative to the base size, so if we write ptr++, then the pointer *ptr* will be shifted forward by 20 bytes

**P**

3000

2500     **arr[5]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3  3000 | 5  3004 | 6  3008 | 7  3012 | 9  3016 |

**ptr**

3000

4500

On dereferencing a pointer expression we get a value pointed to by that pointer expression. Pointer to an array points to an array, so on dereferencing it, we should get the array, and the name of array denotes the base address. So whenever a pointer to an array is dereference, we get the base address of the array to which it points.

In Array access using pointer, Take x[5] be the array with 5 integer element hold base address of array

&x[i] = ith array element address

X[i] = ith array element

&x[i] = ( x + i ) and x[i] = \* ( x + i )

**Q.5)** Explain how pointers to structures can be used

Structure is a user defined data type in C which allows us to combine data of different type together.

We can access a structure member using pointers, of type structure, in the following ways;

**1) Using the arrow operator**: If the members of the structure are public then you can directly access them using the arrow operator ( -> ). If they are private then you can define methods for accessing the values and use pointers to access the methods. The arrow operator can be used to access structure variables as well as methods.

**Syntax:**

Pointer\_Name->member\_Name;

**2) Using Dereferencing operator:**You can also access structure elements using the dereferencing operator on the pointer, which is using an asterisk to dereference the pointer and then using the dot operator to specify the structure element.

**Syntax:**

(\*PointerName).MemberName;

Program:-

#include<stdio.h>

typedef struct student

{

int rno;

char nm[10];

int mrk;

}STU;

Void mian()

{

STU \*q; //pointer to structure

STU p; //structure variable

q=&p;

printf(“\nEnter student data: “);

printf(“\nRoll no: “);

scanf(“%d”,&p.rno);

printf(“\nName: “);

scanf(“%s”,q->nm);

printf(“\nMarks: “);

scanf(“%d”,&q->mrk);

printf(“\nStudent Data \n”);

printf(“\nRoll no :%4d\nName: %-10s\nMarks:%5d”,q->rno,q->nm,q->mrk);

return;

}

If pointer to the structure is declared in the program, then elements of structure can be accessed by **( -> )** operator.

For ex as declared in above example

q->rno;

q->mrk;

q->nm;