PROGRAMMING FOR PROBLEM SOLVING -- PPS

SUBJECT CODE- BTPS-101-18

UNIT – 8
POINTERS

Objectives

- What is meant by pointer is and why it can ne used?
- How to Declare and access a pointer variable
- What is Linked List
- What is Self Referential Structure
- Beyond the syllabus--Activity
- Example: Pointer Arithmetic–

POINTER

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

Declaration of pointer :

syntax : datatype *pointername;

Example :

```
int *ptr;
char *pt;
```

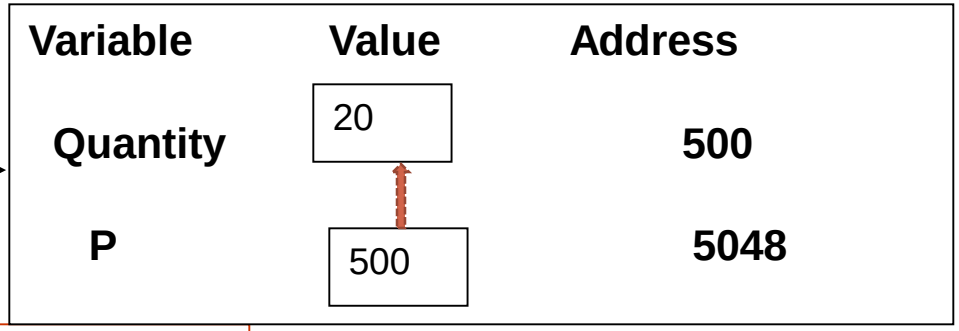
Assigning data to the pointer variable

syntax :

pointervariablename=&variablename;

For Example :

```
int *p,quantity=20;
p=&quantity;
```



For Example :

```
#include<stdio.h>
void main()
{int quantity=20,*p;
p=&quantity;
printf("%u\n",&p);
Printf("%d\n",p);
printf("%d\n",*p);
printf("%d\n",*(&quantity));
}
```

Output:

5048
500
20
20

Why are Pointers Used ?

- To return more than one value from a function
- To pass arrays & strings more conveniently from one function to another
- To manipulate arrays more easily by moving pointers to them, Instead of moving the arrays themselves
- To allocate memory and access it (Dynamic Memory Allocation)
- To create complex data structures such as Linked List, Where one data structure must contain references to other data structures

Advantages:

☺ A pointer enables us to access a variable that is defined outside the function.

☺ Pointers are more efficient in handling the data tables.

☺ Pointers reduce the length and complexity of a program.

☺ They increase the execution speed.

☺ The use of a pointer array to character strings results in saving of data storage space in memory.

☺ The function pointer can be used to call a function

☺ Pointer arrays give a convenient method for storing strings

☺Many of the ‘C’ Built-in functions that work with strings use Pointers

☺It provides a way of accessing a variable without referring to the variable directly

Linked List

- Linked lists are the best and simplest example of a dynamic data structure that uses pointers for its implementation

Advantages:

Linked lists have a few advantages over arrays:

- Items can be added or removed from the middle of the list
- There is no need to define an initial size

Linked List

Disadvantages:

- There is no "random" access - it is impossible to reach the nth item in the array without first iterating over all items up until that item. This means we have to start from the beginning of the list and count how many times we advance in the list until we get to the desired item.
- Dynamic memory allocation and pointers are required, which complicates the code and increases the risk of memory leaks and segment faults.
- Linked lists have a much larger overhead over arrays, since linked list items are dynamically allocated (which is less efficient in memory usage) and each item in the list also must store an additional pointer.

Linked List

- A linked list is a set of dynamically allocated nodes, arranged in such a way that each node contains one value and one pointer.
- The pointer always points to the next member of the list. If the pointer is NULL, then it is the last node in the list.
- A linked list is held using a local pointer variable which points to the first item of the list. If that pointer is also NULL, then the list is considered to be empty.

```
typedef struct node
{ int val; struct node * next; } node_t;
```

Self Referential Structure

A self referential structure is used to create data structures like linked lists, stacks, etc. Following is an example of this kind of structure:

```
struct struct_name
{
datatype datatypename;
struct_name * pointer_name;
};
```

.

Self Referential Structure

A self-referential structure is one of the data structures which refer to the pointer to (points) to another structure of the same type. For example, a linked list is supposed to be a self-referential data structure. The next node of a node is being pointed, which is of the same struct type. For example,

```
typedef struct listnode {
void *data;
struct listnode *next;
} linked_list;
```

In the above example, the listnode is a self-referential structure – because the *next is of the type struct listnode

Self Referential Structure

Example:

```
struct node {
    int data1;
    char data2;
    struct node* link;
};

int main()
{
    struct node ob;
    return 0;
}
```

- *In this example 'link' is a pointer to a structure of type 'node'.*
- *Hence, the structure 'node' is a self-referential structure with 'link' as the referencing pointer.*

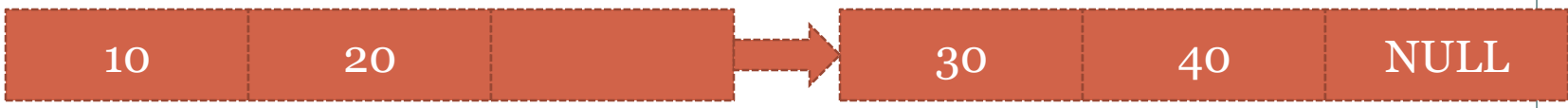
Self Referential Structure

Types of Self Referential Structures

- Self Referential Structure with Single Link
- Self Referential Structure with Multiple Links

Self Referential Structure with Single Link:

- These structures can have only one self-pointer as their member.
- The following example will show us how to connect the objects of a self-referential structure with the single link and access the corresponding data members.



Obj1

Obj2

Example –Self Referential Structure

```
#include <stdio.h>
  struct node {
    int data1;
    int data2;
    struct node* link;
};
  int main()
{     struct node ob1; // Node1
       // Initialization
    ob1.link = NULL;
    ob1.data1 = 10;
    ob1.data2 = 20;
      struct node ob2; // Node2
      // Initialization
    ob2.link = NULL;
    ob2.data1 = 30;
    ob2.data2 = 40;
      // Linking ob1 and ob2
    ob1.link = &ob2;
      // Accessing data members of  ob2 using ob1
    printf("%d", ob1.link->data1);
    printf("\n%d", ob1.link->data2);
    return 0;
}
```

Output

30

40

Beyond The Syllabus – Activity Programs to Practice

Example:1

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n=10;
int *ptr;
ptr=&n;
printf("Value of n is %d",n);
printf("\nAddress of n is %x",&n);
printf("\nAddres of pointer is %x",ptr);
printf("\nvalue stored in pointer is
%d",*ptr);
getch();
}
```

Example 2

```
#include<stdio.h>
#include<stdlib.h>
#define size 10
void main()
{
char name[size];
char *i;
printf("\n Enter your name ");
gets(name);
i=name;
printf("\n Now printing your name is    :");
while(*i != '\0')
{
printf("%c",*i);
i++;
}
}
```

Explain how the variable can be accessed by pointer

```
#include<stdio.h>
#include<conio.h>
void main()
{
int r;
float a,*b;
clrscr();
printf("\n Enter the radius of the circle");
scanf("%d",&r);
a=3.14*r*r;
b=&a;
printf("\n The value of a=%f",a);
printf("\n The value of a=%u",&a);
printf("\n The value of b=%u",b);
printf("\n The value of a=%f",*b);
getch();
}
```

Pointer Arithmetic

- Addition and subtraction are the only operations that can be performed on pointers.
  - Take a look at the following example :

```
int var, *ptr_var;
ptr_var = &var;
var = 500;
ptr_var++ ;
```

    - Let var be an integer type variable having the value 500 and stored at the address 1000.
    - Then ptr_var has the value 1000 stored in it. Since integers are 2 bytes long, after the expression “ptr_var++;” ptr_var will have the value as 1002 and not 1001.

Pointer Increment process example

```
#include<stdio.h>
#include<conio.h>
void main()
{
int *ptr;  //static memory allocation
clrscr();
ptr=(int *) malloc(sizeof(int));
*ptr=100;
printf("\n%u\n",ptr);  //address of ptr
printf("\n%d\n",*ptr);
ptr++;  // increment 2 bytes
*ptr=101;
printf("\n%d\n",*ptr);
free(ptr);
getch();
}

/* Note : int *ptr=100 means 100 is a address
 *ptr=100 or 101  means 100 is a value */
```

POINTS

- Each time a pointer is incremented, it points to the memory location of the next element of its base type.
- Each time it is decremented it points to the location of the previous element.
- All other pointers will increase or decrease depending on the length of the data type they are pointing to.
- Two pointers can be compared in a relational expression provided both the pointers are pointing to variables of the same type.

Increment & Decrement Operations Using Pointer

```
#include<stdio.h>
void main()
{
int i=100,*iptr;
float f=122.354,*fptr;
char c='d',*cptr;
iptr=&i;
fptr=&f;
cptr=&c;
printf("The values of the variables");
printf("\n%d",*iptr);
```

```
printf("\n%f",*fptr);
printf("\n%c",*cptr);
printf("\nStarting Address");
printf("\n%u",iptr);
printf("\n%u",fptr);
printf("\n%u",cptr);
iptr++;
fptr++;
cptr++;
printf("\nPointer Incrementing");
printf("\n%u",iptr);
printf("\n%u",fptr);
printf("\n%u",cptr);
iptr--;
fptr--;
```

```
cptr--;
printf("\nPointer Decrementing");
printf("\n%u",iptr);
printf("\n%u",fptr);
printf("\n%u",cptr);
getch();
}
```

POINTER OPERATORS

& → Returns the memory address of the operand

* → It is the complement of &. It returns the value contained in the memory location pointed to by the pointer variable's value

Write a C program to find the length of the string Using Pointer

```
#include<stdio.h>
#include<string.h>
void main()
{
char *text[20],*pt;
int len;
pt=*text;
printf("Enter the string");
scanf("%s",*text);
while(*pt!='\0')
{
putchar(*pt++);
}
len=pt -*text;
printf("\nLength of the string is %d",len);
}
```

```
/*Add Two Numbers using pointers*/
#include<stdio.h>
void main()
{
int a,b,*c=&a,*d=&b;
printf(“Enter two numbers to be summed");
scanf("%d %d",&a,&b);
printf(“The sum of two numbers=%d",c +
d);
getch();
}
```

```
/* Array Using Pointers */
#include<stdio.h>
#include<conio.h>
void main()
{
int b[100],*iptr;
iptr=&b;
clrscr();
printf("The initial value of iptr is %u\n",iptr);
iptr++;
printf("Incremented value of iptr is %u\n",iptr);
getch();
}
```

POINTERS IN ARRAYS

```
#include<stdio.h>
#include<conio.h>
void main()
{
int arr[]={10,20,30,40};
int *ptr,i;
clrscr();
ptr=arr;  // same as &arr[0];
printf("\n The Result of the array is ");
for(i=0;i<4;i++)
{
printf("%d\n",*ptr);
ptr++;
}
getch();
}
```

Pointers as Function Arguments

When pointers are passed to a function

- The address of the data item is passed and thus the function can freely access the contents of that address from within the function

- In this way, function arguments permit data-items to be altered in the calling routine and the function.

- When the arguments are pointers or arrays, a call by reference is made to the function as opposed to a call by value for the variable arguments.

FUNCTION POINTER EXAMPLE

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

int (* function) (int,int);  /*function pointer prototype */

int addition(int a,int b)
{
return a+b;
}

int subtraction(int a,int b)
{
return a-b;
}
void main()
{
int val;
/* assign the func. addition into the function pointer */
function=addition;
/* Invoke the func. Addition  */
val=(function) (20,100);

printf("\n Addition result =%d",val);

/* assign the function subtraction into the function pointer */
function=subtraction;

/* invoke the func. subtraction & syntax for function pointer call */

val=(function) (200,100);
printf("\nSubtraction result =%d",val);
getch();
}
```

Pointers To Structures

- Pointers to structures are allowed in C, but there are some special aspects to structure pointers that a user of pointers to structures must be aware of.

- The following statement declares ptr as a pointer to data of that type – Struct book *ptr;

How the structure can be accessed by a pointer variable

```
#include<stdio.h>
#include<stdlib.h>
struct student
{
int roll_no;
char name[20];
float marks;
}st;
void main()
{
struct student *ptr;
printf("\n \t Enter the record");
printf("\n Enter the Roll Number");
scanf("%d",&st.roll_no);

printf("\n Enter the Name");
scanf("%s",st.name);

printf("\n Enter the Marks");
scanf("%f",&st.marks);

ptr=&st;
printf("\n display the details using structure variables");

printf( "%d %s %f", st.roll_no, st.name, st.marks);

printf("\n display the details using pointer variables");
printf( "%d %s %f",ptr->roll_no,        ptr->name, ptr->marks);
}

void print_rec(int r,char n[ ],float m)
{
printf("\n You have Entered Following record");
printf("\n %d %s %f",r,n,m);
}
```

THANK YOU