ALGORITHM & DATA STRUCTURE

COMPUTER SCIENCE & IT

COMPUTER SCIENCE & IT

GATE / PSUs

STUDY MATERIAL

ALGORITHM & DATA STRUCTURE

ENGINEERS INSTITUTE OF INDIA

ENGINEERS INSTITUTE OF INDIA

COMPUTER SCIENCE & IT

GATE & PSUs

STUDY MATERIAL

ALGORITHM & DATA STRUCTURE

CONTENT

CHAPTER-1
C POINTERS

INTRODUCTION:

Pointers are frequently used in C, as they offer number of benefit to the programmer. They include

- Pointers are more efficient in handling array and data table.
- Pointer can be used to return multiple values from a function via function argument.
- Pointer permit reference to function and thereby facilitating passing of functions as argument to other functions.
- The use of pointer array to character string result in saving of data storage space in memory.
- Pointer allows C to support dynamic memory management.
- Pointer provides an efficient tool for manipulating dynamic data structure such as structure linked list queue, stacks and trees.
- Pointer reduces length and complexity of program.
- They reduce length and complexity of program.
- They increase the execution speed and thus reduce the program execution time.

Understanding pointers:

Whenever we declare a variable then system allocate somewhere in memory, an appropriate location to hold the value of variable.

Consider the following statement

```
intx  =  80;
x    ← variable
80   ← value
5000 ← address
```

Representation of variable

- Pointer variable is nothing but a variable that contain an address which is a location of another variable in memory.
  For example:

Variable	*Value*	*Address*
x	80	5000
p	5000	5048

Pointer variable

Value of variable P is the address of variable X

- Address of a variable can be determined by & operator in C

  For example: $P = \& x$

  Would assign 5000 to variable P

  & operator can be used with simple variable or an array element. The following are the illegal use of address operator

  - & 125 pointing constant
  - int x [10]

    & x (pointing an array name)
  - & (x + y) (pointing at expression)

It x is an array then expression such as & x [0] and & x (i+3) are valid and represent the address of 0^{th} and $(i+3)^{th}$ element of

- int * P

The declaration causes the compiler to allocate memory location for the pointer variable. Since memory location has not been assigned any value. These locations may contain some unknown value in them. Therefore they point to some unknown location.

$P \quad \boxed{?} \rightarrow ?$

Garbage point to unknown Location

- We can have multiple declaration in same statement

  * int $^{*}p, q, ^{*}r;$

* int $x, {}^{*}p, y;$

$x = 10$;

$p = \&x$

- It is also possible to combine the declaration of data variable and declaration of pointer variable and Initialization of pointer variable in one step.

  int $x, {}^{*}p = \&x$;
- We can also define the pointer variable with initial value of null or zero. We can also assign the constant value to pointer variable.
- Pointer are flexible we can make same pointer to point different data variable in different statement.
- We can also use different pointer to point the same data variable.

- int $x, y, {}^{*}ptr$;

  $ptr = \&x$;

  $y = {}^{*} ptr$ //Assign the value pointed by pointer ptr to y

  $*ptr = 25$; // This statement put the value of 25 at the location whose address is the value of pointer variable; we know the value of pointer variable (ptr) is the address of x.

  Therefore old value of x is replaced by 25.

  It is equivalent to assigning value 25 to x.

Illustration of Pointer Assignments:

Stage	Value in storage		Address
Declaration	*x*	*y*	*ptr*
	4104	4108	4106 ← *Address*
$x = 10$	10		
	4104	4108	4106 ← *Address*
$ptr = \&x$	10		4104
	4104	4108	4106 ← *Address*
$Y = *ptr$	10	10	4104
	4104	4108	4106 ← *Address*
$*ptr = 25$	25	10	4104
	4104	4108	4106 ← *Address*

Chain of pointers:

$$\underset{p_2}{\boxed{address2}} \rightarrow \underset{p_1}{\boxed{Address1}} \rightarrow \underset{Variable}{\boxed{Value}}$$

Pointer p_2 contain the address of pointer p_1 which points to the location that contain the desired value

For example

int * * p_2 ;

- Arithmetic operation on pointers

Multiplication:

$$y = *ptr \times *ptr$$

Multiplication of $*p_1$ *and* $*p_2$

Division:

$$Z = *p_2 / \underbrace{*p_1}$$

Space is necessary /* considered as beginning of comment.

- $p_1 + 4,\ \ p1 - 5,\ \ p_2 - 2,\ p_1 ++,\ -p_2$ all the operation are allowed
- $p_2 > p_1,\quad p_1 == p_2 \quad and \quad p_1 != p_2$ are allowed
- $p_1 - p_2$ is also allowed. If p_1 *and* p_2 both points to same array then $p_1 - p_2$ gives the number of element between p_1 *and* p_2.

  However

  $p_1 / p_2,\quad p_{1*}p_2,\quad p_1 / 3,\quad p_{1*}3\ , p_1 + p_2$ Are not allowed
- $p_1 ++$ Will cause the pointer p_1 to point to next value of its type. suppose p_1 is integer. Pointer with initial value 2800 then after operation $p_1 ++$ value of p_1 will be 2802 not 2801
- A value cannot be assigned to an arbitrary address i.e. $\&x = 10$ is illegal

$$Size\ of\ \text{int} = 2$$
$$char = 1$$
$$float = 4$$
$$size\ of\ pointer = 2$$

- $p = 1000$
- $\&p = 2000$
- $*p = 10$

Then

$$*(\&p) = 1000$$
$$*(*(\&p)) = 10$$

Pointer representation in Array:

1-D: A[3] can be represented by $*[A+3]$

Base or starting address of array

2 D: $a[i][i]=*(*(a+i)+j)$

In 1-D array to get value stored we need one asterisk (pointer notation)

In 2-D array to get value stored in array we need two asterisk.

In 3-D array to get value stored in array we need three asterisk.

P: Pointer to the first row

P+i: pointer to itn row

$*(p+i)$: Pointer to the first element in its row

$*(p+i)+j$: Pointer to j^{th} element in i^{th} row

$*(*(p+i)+j)$: Value stored in cell (i,j) i.e. i^{th} Row j^{th} column

- Pointer to one data type can be used to point another data type. This is possible by type casting
  - Pointer to float number can be used as pointer to an integer
  - Some conversion are required

$$Pi = (\text{int}*)\, pf$$

↑ ↑

Pointer to integer pointer to float no

- Difference between $*p[3]$ and $(*p)[3]$. Since precedence of [] is more than $*p[3]$ declare p as an array of 3 pointer while $(*p)[3]$ declare p as a pointer of an array having 3 elements.

- When an array is passed to a function as an argument, only the address of first element of an array is passed but not the actual value of array element.

For example:

$$\text{int } x[10];$$
$$sort\ (x)$$

Address of x [0] is passed to function sort

- Call by reference OR call by address

main ( )

$$\{\text{int } x;$$
$$x = 20;$$
$$change(\&x)$$
$$pr\,\text{int}\, f\,("\%d \backslash n, x);\}$$

$$\{$$
$$*p = *p + 10;$$
$$\}$$

Output 30

When function change ( ) is called only the address of x is passed to the pointer P not value.

- Parameters are passed by value in c, i.e. values being passed are copied into the parameter of called function at the time when function's invoked.

Pointer notation and their significance:

1. int * p // p is a pointer to an integer quantity.

2. int * $p[10]$

   P is a 10 – element array of pointer to integer quantities.

3. int $(*p)[10]$

   P is a pointer to a 10 – element integer array

4. int * $p[void]$

   P is a function that returns to an integer quantity

5. int $p[char * a]$

   P is a function that accepts an argument which is a pointer to character & return an integer quantity

6. int * $p[char * a]$

   P is a function that accept an argument which is a pointer to a character return a pointer to an integer quantity

7. int $(*p)[char * a]$

   P is a pointer to function that accept an argument which is a pointer to a character return an integer quantity

8. int $((*p)(char * a))[10]$

   P is a function that accept an argument which is a pointer to a character return a pointer to a 10 – element integer array

9. int $p(char(*a)[\ ])$

   P is a function that accepts an argument which is a pointer to a character array returns an integer quantity.

10. int $p(char * a[\ ]);$

    P is a function that accepts an argument which is an array of pointer to characters returns an integer quantity

11. int * $p(char\ a[\ ]);$

    P is a function that accepts an argument which is a character array returns a pointer to an integer quantity

12. int $*p(char\ (*a)[\ ])$;

P is a function that accepts an argument which is a pointer to character array return a pointer to an integer-quantity

13. int $*p(char\ *a[\ ])$

P is a function that accept an argument which is an array of pointer to character return to an integer quantity

14. int $*p(char\ (*a)[\ ])$;

P is a function that accept an argument which is a pointer to a character array return an integer quantity

15. int $(*p)(char\ (*a)[\ ])$

P is a pointer to a function that accepts an argument which is a pointer to a character array return a pointer to an integer array.

16. int $*(*p)char\ *a[\ ]$

P is a ptr to fun! That accepts an argument which is an array of pointer to character return a pointer to an integer quantity.

17. int $*p[10](void)$

P is a 10 element array of a pointer to function, each function return an integer quantity.

18. int $(*p[10])(char\ a)$;

P is a 10 element array of pointer to functions. Each function accepts an argument which is a character & returns an integer quantity

19. int $*(*p[10])(char\ a)$

P is a 10 element array of pointer to functions each function accepts an argument which is a character and return a pointer to an integer quantity

20. int $*(*p[10])(char\ *a)$

P is a 10 element array of a pointer to functions; each function accepts an argument which is a pointer to a character and return a pointer to an integer quantity.

PRACTICES SET

Question: What is O/P of following C program

```
int f(int x, int* py, int** ppz)
{
    int z, y
    ** ppz +=1
    z = ** ppz
    * py + = 2
    y = * py
    x = x+3
    return (x+y+z)
}

Main ( )
{
    int c, *b, **a
    c = 4, b = &c, a = &b
    printf ("%d", f (c, b, a))
}
```

Ans: 19

Explanation

a	*b*	*c*
2000	1000	4
3000	2000	1000

x	*p*4	*ppz*
4	1000	2000
6000	5000	4000

** *ppz* return 4 and ** *ppz* + = 1 return 5

So z = 5

**py* + = 2 *return* 7

So y = 7

As value of x = 4

So x = x+3 return 7

So output is 7+7+5 = 19

Question: What does following program print:

```
# include <studio. h>

Void f(int* p, int* q)

    {  p=q
       *p=z
       }

int i = 0, j =1;
int main ( )
  {  f (&i,& j);
     printf(: %d %d \n", i, j);
     return 0;
     }
```

Ans: 0,2

Explanation:

i	*j*	*p*	*q*
0	1	3000	4000
3000	4000	1000	2000

After $p = q$

p	*q*
4000	4000

So $*p = 2$ *change value of* j *and value of* i *does not effected.*

Question: Consider the following c – program

```
int x

Void Q (int z)

{
    z += x
    printf("%d", z)
}

void p (int *y)

{
    int x = *y + 2
    Q(x)
    *y = x - 1
    printf("%d", x)
}

Main ( )

{
    x = 5
    p(&x)
    printf("%d", x)
}
```

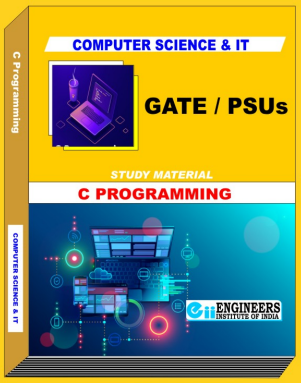
What is value of z, x, x in sequence

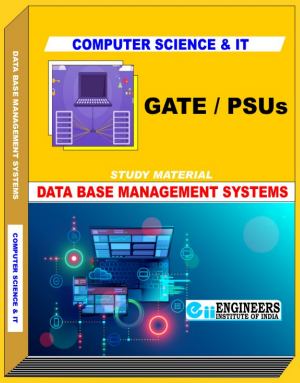
Output

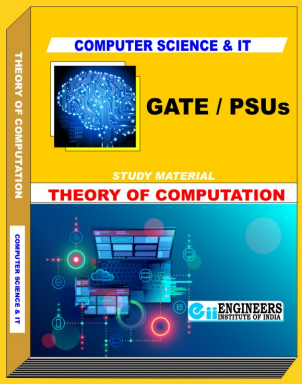
12, 7, 6

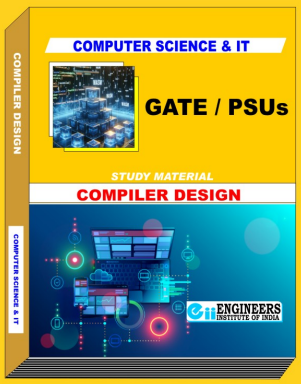
Explanation: variable x is declared globally and it is initialized in main function. int x declared in void p has scope only within the function definition.

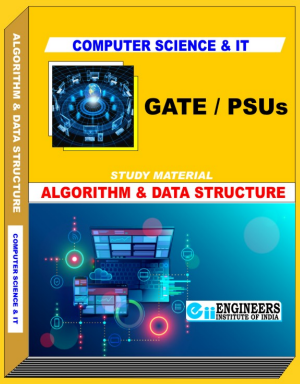
Published Books

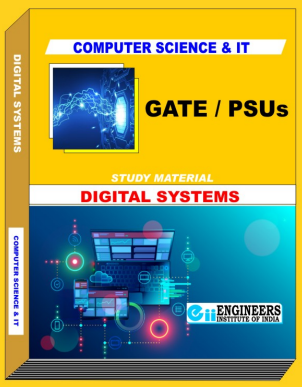


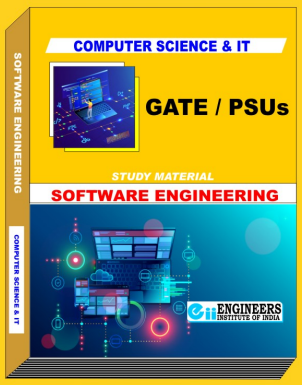


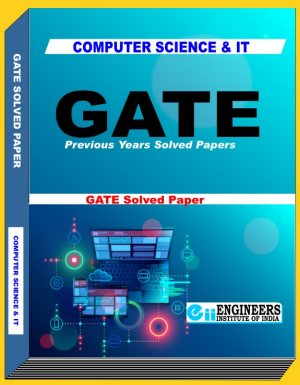


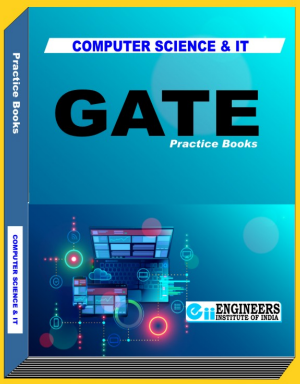












ENGINEERS INSTITUTE OF INDIA

Classroom Batches | Online Classes

Postal Course Classes | Online Test Series

Office: 58B, Kalu Sarai Near Hauz Khas Metro Station New Delhi-16

Helpline: 9990657855 , 9990357855

www.engineersinstitute.com