Pointers (class 3)

05-03-2023

Date.....

* Pointer to Pointer :- (Double Pointer)

we know that pointer is a variable that can contain memory address. The pointer variable takes some spaces in memory & therefore it also has an address. we can store the address of a pointer variable in some other variable, which is known as Pointer to Pointer variable.

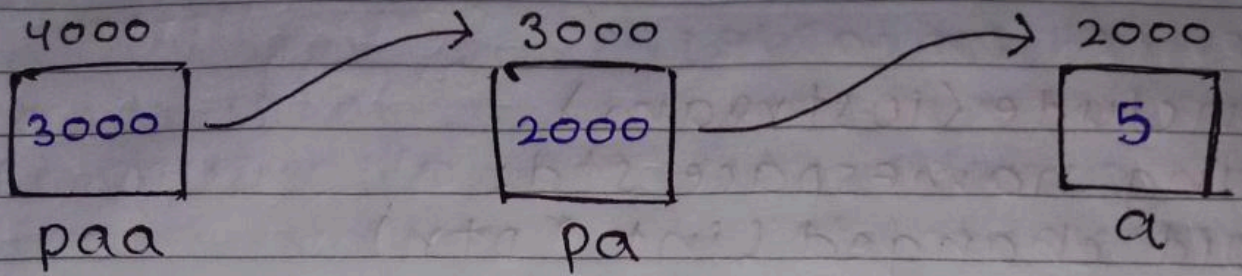
Similarly, we can have a pointer to pointer to pointer variable & this concept can be extended to any limit, but in practice only pointer to pointer is used. Pointer to Pointer is generally used while passing pointer variables to functions,

```
int a = 5;
int *pa = &a;
int **paa = &pa;
```

type of variable 'a' is int

type of variable 'pa' is int * or pointer to int

type of variable 'paa' is int ** or pointer to pointer to int.



'pa' is a pointer variable, which contains the address of a and paa is a pointer to pointer variable, which contains the address of pointer variable pa.

we Know that *pa gives value of a, similarly *paa will give the value of pa.
what **paa gives?

**paa
*(*paa) → Since *paa gives pa
*pa → *pa gives a.
a

Means, *paa gives value of a.
Table,

value of a	a	*pa	**paa	5
Address of a	&a	pa	*paa	2000
value of pa	&a	pa	*paa	2000
Address of pa		&pa	paa	3000
value of paa		&pa	paa	3000
Address of paa			&paa	4000

Date.....

code:-

```
#include<iostream>
using namespace std;
void changed(int *ptr)
{
    ptr = ptr + 1;
}
int main()
{
    int a = 5;
    int *p = &a;
    cout << "Before :" << endl;
    cout << a;
    cout << &a;
    cout << *p;

    changed(p);

    cout << "After :" << endl;
    cout << a;
    cout << &a;
    cout << *p;

    return 0;
}
```

Date.....

Here we created a variable a & a pointer variable which contains address of a.
First, we print value of a, address of a & value at p. Its, 5, 104 & 5, respectively.
Then we pass our pointer to our changed function. In the function we change the value of our pointer & increment its value by 1.
Again, we print value of a, address of a & value at p. Its, 5, 104 & 5, respectively.

It is because in changed function, the pointer is passed as value.
So any change in the changed function will not reflect to the actual values.
If we want to change the value of a in the function, then we have to change at the value at pointer, i.e.,

```
void changed (int *ptr)
{
  *ptr = *ptr + 1;
}
```

↳ this will directly change the value of a.

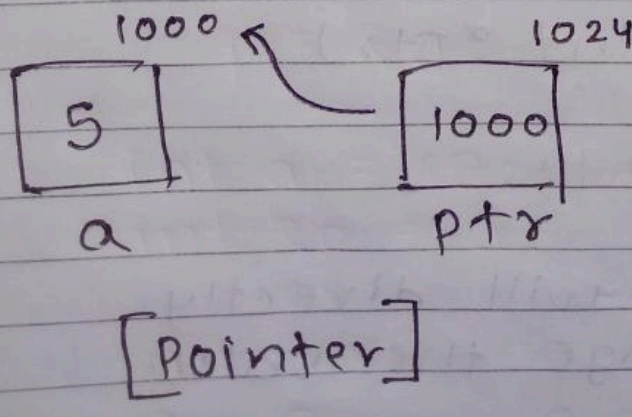
Date.....

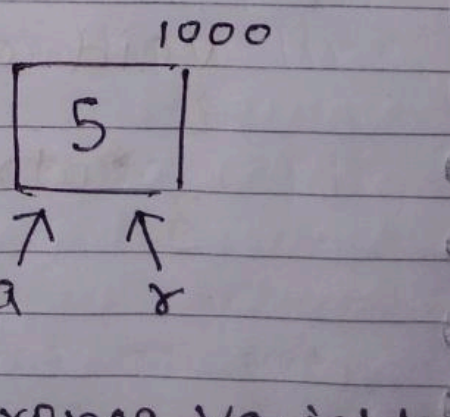
Reference variables (same memory location, different names)

C++, along with pointers, also support reference variables.
On the surface, both reference variable & pointers are very similar as both are used to have one variable provide access to another.

→ A reference variable provides an alias (alternate name) for a previ-ous defined variable. A reference variable, like a pointer, is also imple-mented by storing the address of an object.

→ A pointer is a variable that holds the memory address of another variable. A pointer needs to be dere-ferenced with the * operator to access the memory location it points to.





Date.....

For example :- If we make the varia-ble 'sum' & a reference to that variable 'total', then 'sum' & 'total' can be used interchangeably to represent that variable.
Reference variable is created as follow:

```
int sum = 100;
int &total = sum;
```

↳ reference variable to variable 'sum'.

'sum' is a int type variable that has already been declared.
'total' is the alternate name declared to represent the variable 'sum'.
Both the variables refer to the same data object in the memory.
Now,

```
cout << sum;
cout << total;
```

} Both print the value 100.

The statement,

```
sum = sum + 50;
```

↳ this will change the value of both sum & total to 150.

```
total = 0;
```

↓
this will change the value of both the variables to 0.

```
int n = 5;  → integer variable.
int *p = &n;
int &r = n;
```

pointer to integer variable

reference to integer variable.

A reference variable must be initialized at the time of declaration, as a reference to a particular variable, & this reference cannot be changed.
Reference variable cannot be null.
A reference variable is implemented as a constant pointer to the variable.

* Difference between Pointer & Reference variable :-

① initialization :-

```
int a = 10;
int *p = &a;
   or
int *p;
p = &a;
```

* Pointers we can declare & initialize pointer at same step or into multiple line.

```
int a = 10;
int &p = a;
   or
int &p;
p = a;
```

* Reference Variable

✗ this is incorrect as we should declare & initialize reference as single step.

Spiral

Teacher's Sign .....

(2) Reassignment :-

```
int a = 5;
int b = 6;
int *p;
p = &a;
p = &b;
```

* Pointers.
A pointer can be re-assigned. This property is useful for the implementation of Data Structures like linked list, tree, etc.

```
int a = 5;
int b = 6;
int &p = a;
```

* Reference variable

```
int &p = b;
```

→ this line throws an error of "multiple declaration not allowed".

```
int &q = p;
```

↓

this is a valid statement.

(3) Memory address :-

A pointer has its own memory addre-ss & size on the stack, whereas a reference shares the same memory address with the original variable but also takes up some space on the stack.

```
int &p = a;
cout << &p << endl << &a;
```

Spiral

Teacher's Sign .....

Date.....

(4) Null value :-

A pointer can be assigned Null directly whereas a reference cannot be.
The constrain associated with referen -ce variable is (no null, no reassig -nment).

(5) Indirection :-

we can have pointer to pointer, i.e; double pointer, offering extra level of indirection, whereas reference only offer one level of indirection.

```
int a = 10;
int *p;
int **q;
p = &a;
q = &p;
```

* Pointers.
extra level of indirecti -on means, we can create double pointer.

```
int &p = a;
int &&q = p;
```

* Reference variables.

↳ this gives an error because we are trying to create reference to a reference varia -ble, which is not possible.

Date.....

(6) Arithmetic Operations :-

various arithmetic operations can be performed on pointers, whereas there is no such thing called Reference arithmetic.

—X——X——X——X——X—

* Pass Reference variable in function :-

```
void solve (int &val)
{
   val++;
}

int main()
{
   int a = 5;
   solve (a);
   cout << a;
}
```

this will increment the value of 'a' because it is pass by reference.

```
void solve (int val)
{
   val++;
}
```

↓ this will not increment the value of 'a', because it is pass by value.

Teacher's Sign.....

Date.....

* How to pass pointer as a reference to a function :-

```
void solve (int *p)
{
    p = p + 1;
}
int main ()
{
    int a = 5;
    int *p = &a;
    cout << "before : " << p;
    solve (p);
    cout << "after : " << p;
    return 0;
}
```

→ this will not change actual value of 'a', because it is a pointer & pointer always pass by value.

To solve a pointer by reference in a function is :-

```
void solve ( int *&p)
{
    p = p + 1;
}
```

↳ pass by reference.

* when to use what ?

The performance are exactly the same as reference are implemented internally as pointers.

use reference variables

- In function parameters & return types.

use pointers

- If pointer arithmetic or passing a null pointer is needed. For example, for arrays (note that accessing an array is implemented using pointer arithmetic).
- To implement Data structures like a linked list, a tree, etc, & their algorithms. This is so because, in order to point to different cells, we have to use the concept of pointer.