Www.cgpabooster.in

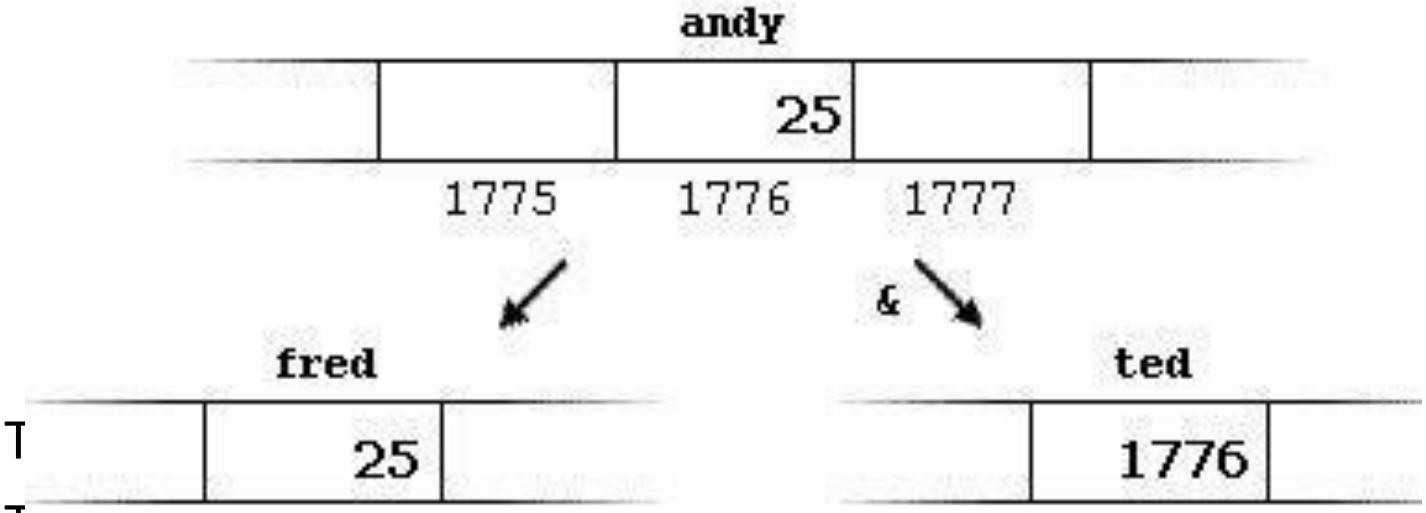
POINTER (IN C/C++)

THE ARC.

What is a pointer?

Variable in a program is something with a name, the value of which can vary. The way the compiler and linker handles this is that it assigns a specific block of memory within the computer to hold the value of that variable.

THE ARC.

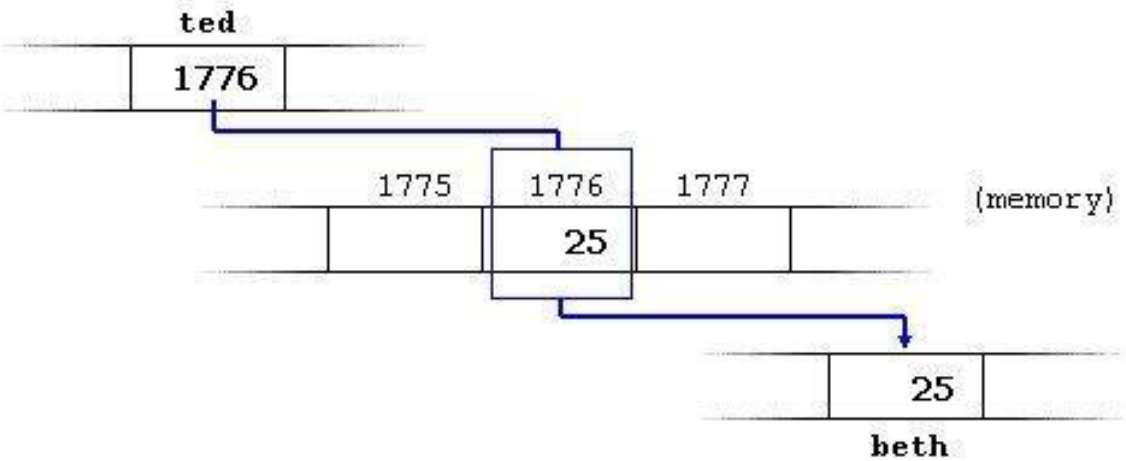


- T
- The right side is the address of that memory

Dereferencing:

- int bar = *foo_ptr;
- *foo_ptr = 42; // set foo to 42 which is also effect bar = 42

```
beth = *ted;
```



- contain in that is 25 which is what we need.

Differences between & and *

& is the reference operator and can be read as "address of“

* is the dereference operator and can be read as "value pointed by"

THE ARC.

A variable referenced with & can be dereferenced with *.

- Andy = 25;
- Ted = &andy;

  All expressions below are true:
- andy == 25 // true
- &andy == 1776 // true
- ted == 1776 // true
- *ted == 25 // true

THE ARC.

How to declare pointer?

- Type + “*” + name of variable.
- Example: int * number;
- char * c;
-
- number or c is a variable is called a ***pointer variable***

How to use pointer?

- int foo;
- int *foo_ptr = &foo;
- *foo_ptr* is declared as a pointer to int. We have initialized it to point to *foo*.
- *foo* occupies some memory. Its location in memory is called its address. `&foo` is the address of *foo*

Assignment and pointer:

- int *foo_pr = 5; // wrong

- int foo = 5;
- int *foo_pr = &foo; // correct way

Change the pointer to the next memory block:

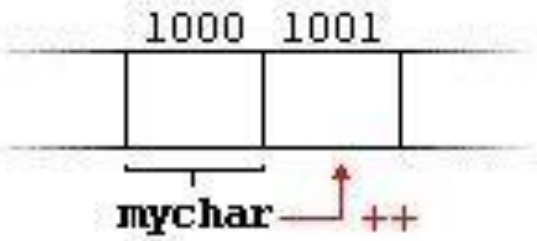
- int foo = 5;
- int *foo_pr = &foo;
- foo_pr ++;

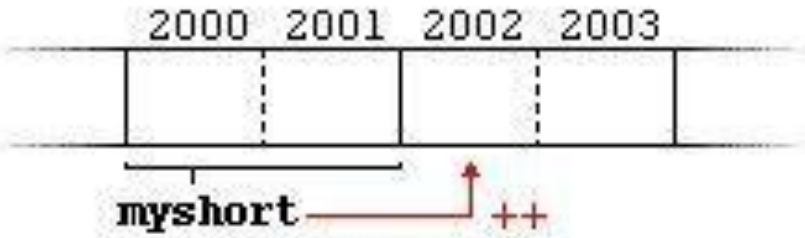
Pointer arithmetics

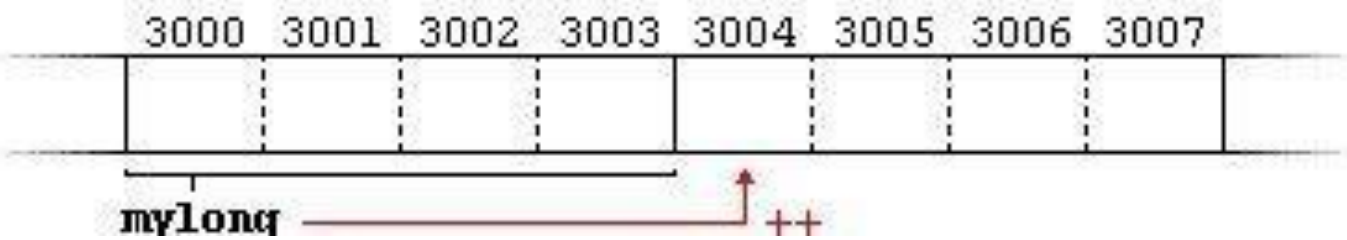
- char *mychar; // sizeof 1 byte
- short *myshort; // sizeof 2 bytes
- long *mylong; // sizeof 4 byts

- mychar++; // increase by 1 byte
- myshort++; // increase by 2 bytes
- mylong++; // increase by 4 bytes

THE ARC.


1000 1001
mychar
++


2000 2001 2002 2003
myshort
++


3000 3001 3002 3003 3004 3005 3006 3007
mylong
++

THE ARC.

Increase pointer is differentfrom increase the dereference

- *P++; // unary operation: go to the address of the pointer then increase its address and return a value
- (*P)++; // get the value from the address of p then increase the value by 1

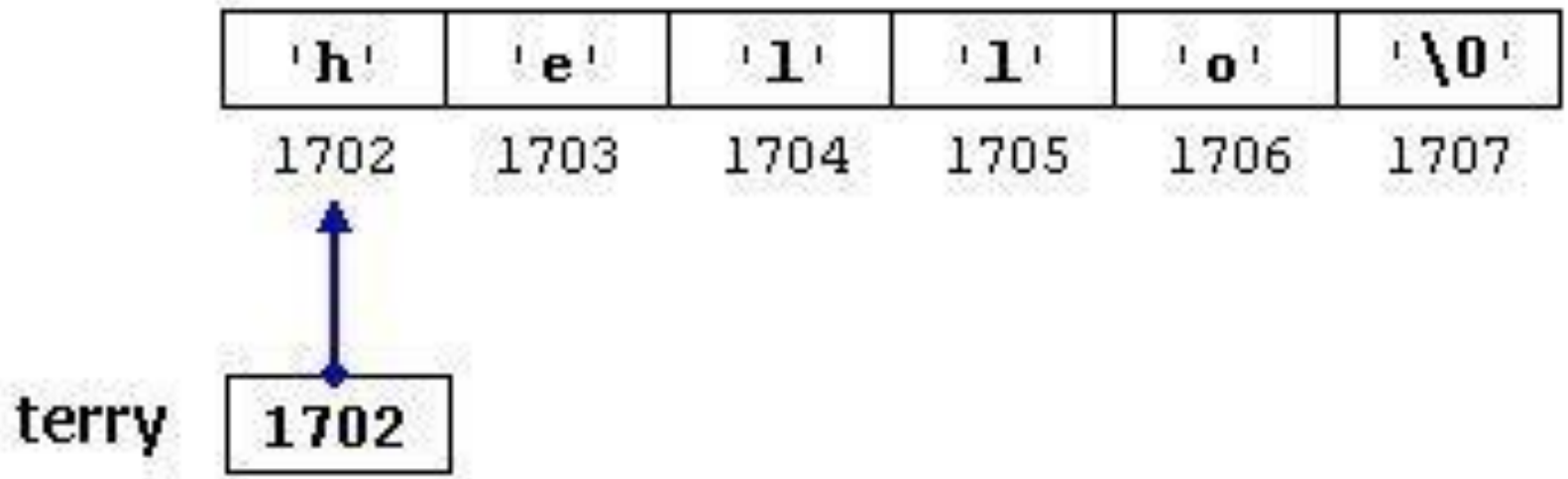
THE ARC.

Arrays:

- int array[] = {45,46,47};
- we can call the first element in the array by saying: *array or array[0].
- Also the second element would be call: *(array +1) or array[1]

Array of character pointer

char * terry = "hello";



Example:

```
char *p1 = &str1[0], *p2 = &str2[0];
while(1) {
    if(*p1 != *p2)
        return *p1 - *p2;
    If(*p1 == '\0' || *p2 == '\0')
        return 0; p1++; p2++;
        }
```

Pointer

→ A pointer is variable which stores the address of another variable.

Syntax: data_type *ptr_name;

eg → int var = 10
int *p;
p = &var → address of var.

→ P is the pointer that stores the address of variable var. The data types of pointer p and variable var should match because an integer pointer can only hold the address of integer variable.

int x = 10;
float y = 2.0;
int *pn = &y; → Invalid {diff both have diff datatype}
↓ ↓
int float
int *ptr = &x; → valid

→ Any number of pointer can point to the same address.

eg. int x = 12
int *p1 = &x, *p2 = &x, *p3 = &x;
↓
All the three pointers are pointing towards x.

→ Memory taken by any kind of pointer (i.e int, float, char float...) is always equivalent to the memory taken by unsigned integer, as pointer will always store address of a variable (which is always unsigned integer), so the type of pointer will not make any difference.

```
#include<iostream>
using namespace std;

int main() {
  int a=3;
  int *b;
  b = &a;
      ↳ (Address of) operator.

  Cout<<" the address of a is"<<&a<<endl;
  Cout<<" " " " " "<<b<<endl;

  * ---> (value at) Deference operator.
  Cout<<" the value at address b is"<<*b<<endl;

  * ---> pointer to pointer.
  int ** c = &b;
    cout<<"the add of b is"<<&b<<endl;
    Cout<<" " " " " "<<c<<endl;
    Cout<<"the value at address c is"<<*c<<endl;
  Cout<<" the value at address value at(value at(c)) is"<<
                                    **c<<endl;

  return 0;
  }
```

Types of Pointer

(I) Null Pointer :

A null pointer is a pointer that does not point to any memory location. It is used to intialize a pointer variable when the pointer does not point to a valid memory address.

Note : It is invalid to dereference a null pointer.

eg

```
#include<iostream>
using namespace std;
int main()
{
    int *ptr=NULL
    int a=10;
    Cout << ptr; // 0 will be displayed.
  Cout<< *ptr; // Invalid (deferencing) as ptr is
                                null pointer.
    ptr = &a;
  Cout<< *ptr; //Now it is allowed, as Null pointer
                has starting pointing somewhere.
  return 0;
}
```

(II) Wild Pointer :

→ Pointer which are not intialized during its defination holding some Junk value (no garbage address) are wild pointer.

→ Every pointer when it is not intialized is defined as wild pointer.

→ As pointer get initialized, start pointer to some variable is defined as pointer, not a wild one.

eg

```
#include<iostream>
using namespace std;
int main()
{
    int *ptr;   // wild pointer.
    int a=10;
  Cout<<ptr;   // gives garbage address value
  Cout<<*ptr;  // gives garbage value stored in the garbage address.
  ptr=&a;  //.
        ↳ Now pointer is not wild pointer.
  Cout<<*ptr;
    return 0;
}
```

(iii) Void pointer:

→ Is a pointer that can hold the address of different data types at different time also called generic pointer.

→ Syntax → void *pointer-name;

→ Here, the void is keyword which can point to any value of any data type.

→ But before accessing the value through generic pointer by dereferencing it, it must be properly typecasted.

eg

```
#include<iostream>
using namespace std;
int main()
{
    int x=10;
    char ch = 'A';
    void *gp;
    gp=&x;
    cout << *(int*)gp);
    gp=&ch;
    cout << *(char*)gp);
    return 0;
}
```

(IV) A Constant Pointer :

→ A Constant pointer, ptr is a pointer that is initialized with an address and cannot point to anything else. But we can use ptr to change the contents of variables pointing to

eg ;

```
int value = 22;
int *const ptr = &value;
```

(V) Dangling pointer :

→ It is a type of pointer which point towards such as memory location which is already deleated / or deallocated.

→ It is a problem associated with pointer, where in a pointer is unnecessarily pointing towards deleated memory location.

→ It can be resolved through assigning Null address once, the memory has been deallocated.

```
#include<iostream>
~~int~~ ~~ma~~ using namespace std;
int main() {

    int *ptr;
    {
        int val=23;
        ptr =&val;
        Cout<< *ptr;      //23 is printed.
        Cout<<ptr;  //address of val is printed.
    }

    ~~Print~~ Cout <<ptr; //same address is printed, even
                val is destroyed, hence ptr is dangling
                pointer.
        ptr = Null; //Solution
        Cout<< ptr; //Now ptr is not a dangelling
                pointer So address value is printed.

    return 0;
    }.
```