# UNIT-5

# Files & Pointers

# BCA-2nd Sem

# Basic Input /Output

| stream | description |
| --- | --- |
| cin | standard input stream |
| cout | standard output stream |
| cerr | standard error (output) stream |
| clog | standard logging(output) stream |

# Standard output (cout)

```
cout << "Output sentence";        // prints Output
                                   sentence on screen
 cout << 120;                      // prints number 120 on
    screen
 cout << x;                        // prints the value of x
    on screen


    cout << "Hello";               // prints Hello
    cout << Hello;                // prints the content of
                                   variable Hello
```

# Standard input (cin)

```
int age;
cin >> age;
```

**i/o example :**

```
#include <iostream>
 using namespace std;
 int main ()
{
 int i;
cout << "Please enter an integer value: ";
cin >> i;
```

# Standard input (cin)

```
cout << "The value you entered is " << i;
cout << " and its double is " << i*2 << ".\n";return0;
}
```

**O/P:-**

Please enter an integer value: 702
The value you entered is 702 and its double is 1404.

# Read/write modes in C++.

## Reading A File

```
#include <fstream.h>
void main() {
 ifstream OpenFile("cpp-home.txt");
 char ch;
 while(!OpenFile.eof())
 {
 OpenFile.get(ch);
 cout << ch;
 }
 OpenFile.close(); }
```

# Read/write modes in C++.

**Writing A File**

```
#include <stdio.h>
 #include <string.h>
 int main(int argc, char * argv[])
{
const char * filename="test.txt";
const char * mytext="Once upon a time there were three bears.";
int byteswritten=0;   FILE * ft= fopen(filename, "wb") ;
```

# Read/write modes in C++.

```
   if (ft)
   {
    fwrite(mytext,sizeof(char),strlen(mytext), ft) ;
    fclose( ft ) ;
   }
    printf("len of mytext = %i ",strlen(mytext)) ;
    return 0;
}
```

# Declarations, Referencing and dereferencing

```
#include<iostream>
 using namespace std;
 void main()
 {
 int x;
 int *ptr_p; x = 5;
 ptr_p = &x;
 cout << *ptr_p;
}
```

In the example above we used ampersand sign (&). This sign is called the reference operator.

# Reference and dereference operators

- In the example above we used ampersand sign (&). This sign is called the reference operator.

- If the reference operator is used you will get the "address of" a variable.

- In the example we said: ptr_p = &x;. In words: store the address of the variable x in the pointer ptr_p.

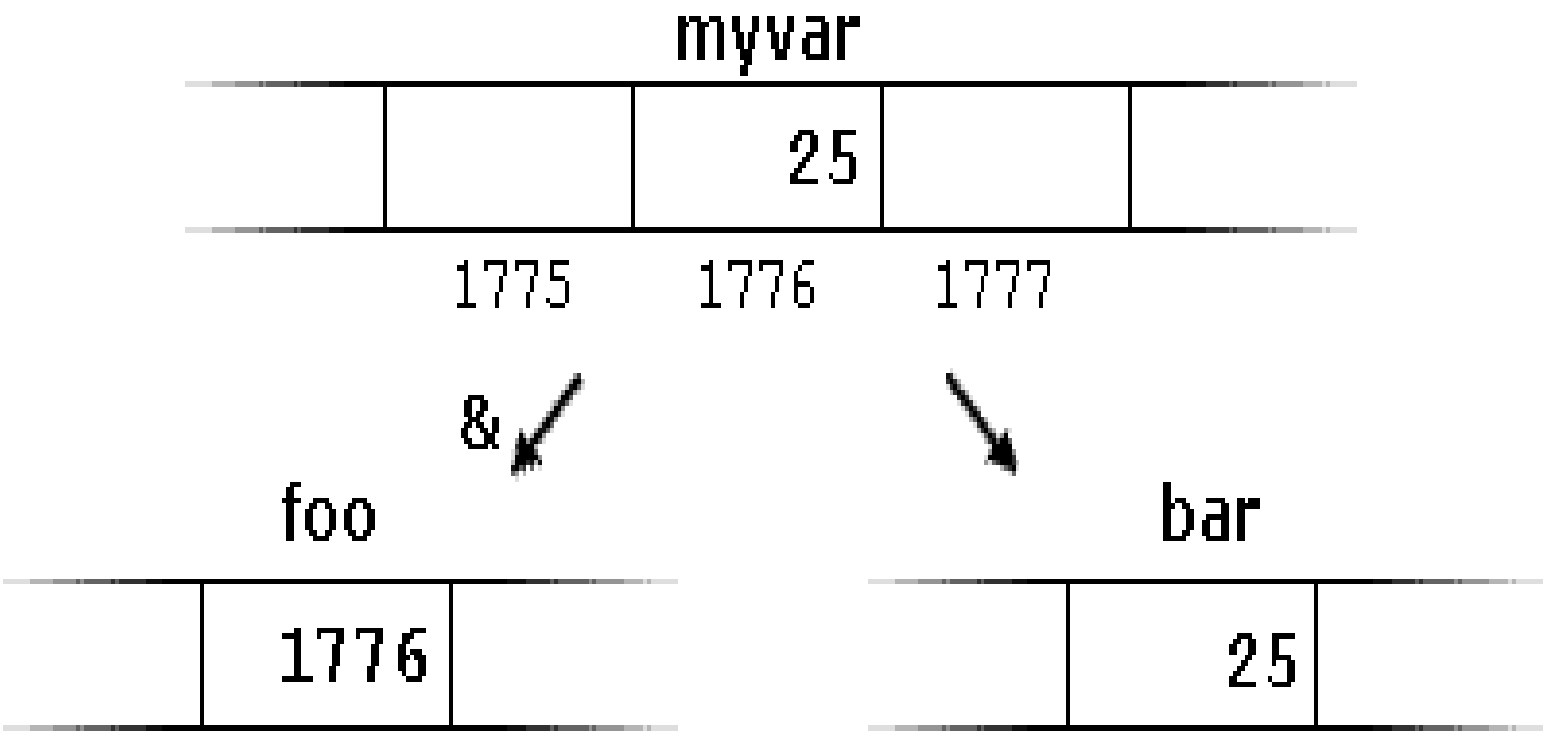
# Reference and dereference operators

- We also used the asterisk sign (*) in the cout statement. This sign is called the dereference operator.

- If the dereference operator is used you will get the “value pointed by” a pointer.

- So cout << *ptr_p;. In words: print (or put into the stream) the value pointed by ptr_p. (It will print the contents of integer x.)

# Reference and dereference operators



# Reference and dereference operators

The address of a variable can be obtained by preceding the name of a variable with an ampersand sign (&), known as reference operator, and which can be literally translated as "address of".

```
foo = &myvar;
myvar = 25;
foo = &myvar;
bar = myvar;
```

# Reference operator

1.First, we have assigned the value 25 to myvar (a variable whose address in memory we assumed to be 1776).

2.The second statement assigns foo the address of myvar, which we have assumed to be 1776.

3.Finally, the third statement, assigns the value contained in myvar to bar. This is a standard assignment operation, as already done many times in earlier.

# Reference operator

4.The main difference between the second and third statements is the appearance of the reference operator (&).

5.The variable that stores the address of another variable (like foo in the previous example) is what in C++ is called a pointer.

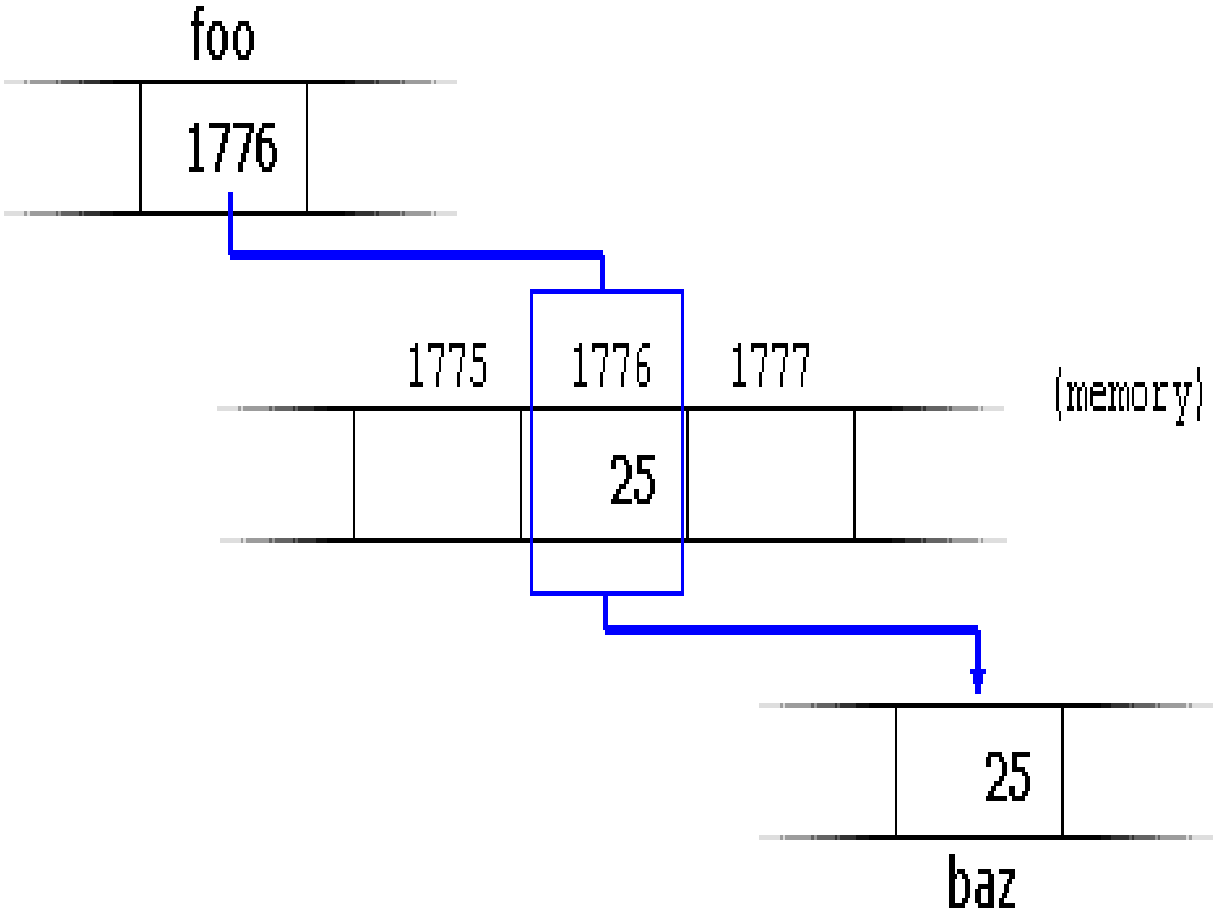
# Dereference operators

- An interesting property of pointers is that they can be used to access the variable they point to directly. This is done by preceding the pointer name with the dereference operator (*). The operator itself can be read as "value pointed toby".

```
baz = *foo;
```

- This could be read as: "baz equal to value pointed to by foo", and the statement would actually assign the value25 to baz, since foo is 1776, and the value pointed to by 1776 (following the example above) would be 25.

# Dereference operators



# Dereference operators

- It is important to clearly differentiate that foo refers to the value 1776, while *foo (with an asterisk * preceding the identifier) refers to the value stored at address 1776, which in this case is 25.

- Notice the difference of including or not including the dereference operator (I have added an explanatory comment of how each of these two expressions could be read):

# Dereference operators

- & is the *reference operator*, and can be read as "address of“
- * is the *dereference operator*, and can be read as "value pointed to by"
-

# Passing Pointer to functions,

- To call the function pointed to by a function pointer, you treat the function pointer as though it were the name of the function you wish to call.

- The act of calling it performs the dereference; there's no need to do it yourself:

# Passing Pointer to functions,

```
#include <stdio.h>
 void my_int_func(int x)
 {
    printf( "%d\n", x );
 }
int main()
 {
 void (*foo)(int);
foo = &my_int_func;
 foo( 2 );
 (*foo)( 2 );
 return 0; }
```

# Pointer to Arrays

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main ()
{
int var[MAX] = {10, 100, 200};
int *ptr[MAX];
for (int i = 0; i < MAX; i++)
{
```

# Pointer to Arrays

```
ptr[i] = &var[i];
}
for (int i = 0; i < MAX; i++)
{
 cout << "Value of var[" << i << "] = ";
cout << *ptr[i] << endl;
}return0;
}
```

# Pointer to Arrays

**OUTPUT:**

Value of var[0] = 10

Value of var[1] = 100

Value of var[2] = 200

# REFRENCES


- Learn Programming in C++ By Anshuman Sharma, Anurag Gupta, Dr.Hardeep Singh, Vikram Sharma