**Q : - What’s a const variable ??**

**Q :- Explain its behavior with variable , function , class ,object ??**

**Q : - Explain its Scope and how can you extend its scope??**

**Q :- Does it have default value?? Storage location??**

**Q:- what’s mutable ?? How we can use it?? Explain issue with it ?? explain scope??default value?? Storage location??**

**Q:- Explain volatile variable ?? Give example?? Explain scope?? Storage location??**

**Parameter :- one [ VARIABLE MEMBER ]**

**Q:- Can we Compile this ???**

#include<iostream>

using namespace std;

int main()

{

const int a;

return 0;

}

Reason s :- This will not compile because **you have not defined the const variable**.

**Q : -How to declare a const variable in class ??**

$ **1.1**

#include<iostream>

using namespace std;

class aa

{

const int a;

public:

aa():a(10){};

void disp()

{

cout<<a;

}

};

int main()

{

aa obj;

obj.disp();

return 0;

}

Where **const int a is private variable** and **declaration in in public section**.

**$ 1.2**

#include<iostream>

using namespace std;

class aa

{

public:

const int a;

aa():a(10){};

void disp();

};

void aa::disp()

{

cout<<a;

}

int main()

{

aa obj;

obj.disp();

return 0;

}

Where **const int a is public variable** and **declaration in in public section**.

$ 2.1 : - Important:- **here c++ basice rules are by passed**

#include<iostream>

using namespace std;

class aa

{

public:

static const int a=10; // you have declared the variable without function

aa(){};

void disp();

};

void aa::disp()

{

cout<<a;

}

int main()

{

aa obj;

obj.disp();

return 0 ;

}

$ 2.2

**#include<iostream>**

**using namespace std;**

**class aa**

**{**

**public:**

**static const int a;**

**aa(){};**

**void disp();**

**};**

**const int aa::a=10;**

**void aa::disp()**

**{**

**cout<<a;**

**}**

**int main()**

**{**

**aa obj;**

**obj.disp();**

**return 0;**

**}**

$3.1

In-Valid way of declaration

#include<iostream>

using namespace std;

class aa

{

public:

static const int a;

aa(){};

void disp();

};

static const int aa::a=10;

void aa::disp()

{

cout<<a;

}

int main()

{

aa obj;

obj.disp();

return 0;

}

**Parameter :- three [ Class ]**

Its gives error message

**“ qualifiers can only be specified for objects and functions “**

Example :-

Const class aa

{

};

**Parameter :- Four [ Class Object ]**

What will happen if you defined a **class object as static** ??

class Something

{

public:

int m\_nValue;

Something() { m\_nValue = 0; }

void ResetValue() { m\_nValue = 0; }

void SetValue(int nValue) { m\_nValue = nValue; }

int GetValue() { return m\_nValue; }

};

int main()

{

const Something cSomething; // calls default constructor

cSomething.m\_nValue = 5; // violates const

cSomething.ResetValue(); // violates const

cSomething.SetValue(5); // violates const

return 0;

}

**Scope** :-

Its scope is local . i.e –

$. 4.1

if define in function its local to function

----------------------------------

void func()

{

Const int a=10;;

}

$. 4.2

if define in file its local to file.

---------------------------------------------

Const int a=10;

Void func()

{

Cout<<a;

}

Void hello()

{

Cout<<a;

}

$ 4.3

Its can be make global by putting extern keyword behind the declarination

Hello.h

------------

**#ifndef** MYHELLO\_H\_

**#define** MYHELLO\_H\_

**extern** **const** **int** a;

**extern** **int** b;

**void** **hello**();

**#endif** /\* MYHELLO\_H\_ \*/

**Bcd.cpp**

**------------**

**#include**<iostream>

**#include** "myhello.h"

**using** **namespace** std;

**const** **int** a=10;

**int** b=20;

**void** **hello**()

{

cout<<"\n hello()"<<a;

cout<<"\n hello()"<<b;

cout<<"hello called";

}

Abc.cpp

**#include**<iostream>

**#include** "myhello.h"

**using** **namespace** std;

**class** aa

{

**public**:

**void** **disp**()

{

cout<<"\n disp() -- "<<a;

}

};

**int** **main**()

{

aa obj;

obj.disp();

hello();

**return** 0;

}

Default value :- No default value

Where does constant store in memeory???

Checkl

Int a const;

**Mutable** : - **The mutable storage class specifier in C++ (or use of mutable keyword in C++)**  
auto, register, static and extern are the storage class specifiers in C. typedef is also considered as a storage class specifier in C. C++ also supports all these storage class specifiers. In addition to this C++, adds one important storage class specifier whose name is mutable.

Actually when you create an **const object of** class and you want to update any variable value within the object then you will used **Mutable Keyword.**

# “Mutable is a qualifiers and storage type in c++”

**Example**

**----------------**

**#include** <iostream>

**using** std::cout;

**class** Test {

**public**:

**int** x;

**mutable** **int** y;

**Test**() { x = 4; y = 10; }

};

**int** **main**()

{

**const** Test t1;

t1.y = 20;

cout << t1.y;

**return** 0;

}

**Issue with mutable variable** : - You cann’t update a mutable variable via function but updating it directly possible because when you try to update its via function . this pointer is passed to function and because the object is const it show the following error “ passing 'const Test' as 'this' argument of 'void Test::disp(int)' discards qualifiers [-fpermissive] abc.cpp “

**#include** <iostream>

**using** **namespace** std;

**class** Test

{

**public**:

**mutable** **int** y;

**Test**(**int** a)

{

y = a;

cout<<y;

}

**void** **disp**(**int**);

**void** **show**();

};

**void** **Test::show**()

{

cout<<y;

}

**void** **Test::disp**(**int** a)

{

y=a;

cout<<y;

}

**int** **main**()

{

**const** Test t1(10);

t1.y = 20;

cout<<t1.y;

t1.disp(30);

t1.show();

**return** 0;

}

Scope :- Similar to variable

Default Value :- no default value

**Volatile:- volatile simple tell the complier not to apply optimization where ever the variable followed by volatile keyword is used because its variable might be modified by some other source that’s not know to complier. Other simple words , I ts simple tell complier not to apply** optimization to the code.

**“** volatile is a hint to the implementation to **avoid aggressive optimization involving the object** because the value of the object might be changed by means undetectable by an implementation. **“**

# “Volatile is a qualifiers and storage type in c++”

**Explanation:-** Consider this code,

int some\_int = 100;

while(some\_int == 100)

{

//your code

}

When this program gets compiled, the compiler may optimize this code, if it finds that the program **never** ever makes any attempt to change the value of some\_int, so it may be tempted to optimize the while loop by changing it from while(some\_int == 100) to simply while(true) so that the execution could be fast (since the condition in while loop appears to be true always). *(if the compiler doesn't optimize it, then it has to fetch the value of some\_int (if it's not loaded on a register) and compare it with 100, each time which obviously is a little bit slow.)*

However, sometimes, optimization (of some parts of your program) may be **undesirable**, because it may be that someone else is changing the value of some\_int from **outside the program which compiler is not aware of**, since it can't see it; but it's how you've designed it. In that case, compiler's optimization would **not** produce the desired result!

So, to ensure the desired result, you need to somehow stop the compiler from optimizing the while loop. That is where the volatile keyword plays it's role. All you need to do is this,

volatile int some\_int = 100; //note the 'volatile' qualifier now!

Links :- http://en.wikipedia.org/wiki/Volatile\_%28computer\_programming%29

**Example :-**

**Actual Code**

**=========**

static int foo;

void bar(void) {

foo = 0;

while (foo != 255)

;

}

Complier optimized

===================

void bar\_optimized(void) {

foo = 0;

while (true)

;

}

To Stop optimization

==========================

static volatile int foo;

void bar (void) {

foo = 0;

while (foo != 255)

;

}

Scope :- Similar to variable

Default Value :- no default value