|  |
| --- |
| Question 1 |

Which of the following is true about this pointer?

|  |  |
| --- | --- |
| A | It is passed as a hidden argument to all function calls |
| B | It is passed as a hidden argument to all non-static function calls |
| C | It is passed as a hidden argument to all static functions |
| D | None of the above |

Explanation:

The ‘this’ pointer is passed as a hidden argument to all non-static member function calls and is available as a local variable within the body of all non-static functions. ‘this’ pointer is a constant pointer that holds the memory address of the current object. ‘this’ pointer is not available in static member functions as static member functions can be called without any object (with class name).

|  |
| --- |
| Question 2  Correct |

What is the use of this pointer?

|  |  |
| --- | --- |
| A | When local variable’s name is same as member’s name, we can access member using this pointer. |
| B | To return reference to the calling object |
| C | Can be used for chained function calls on an object |
| D | All of the above |

Explanation:

See following example for first use.

/\* local variable is same as a member's name \*/

class Test

{

private:

int x;

public:

void setX (int x)

{

// The 'this' pointer is used to retrieve the object's x

// hidden by the local variable 'x'

this->x = x;

}

void print() { cout << "x = " << x << endl; }

};

And following example for second and third point.

#include

using namespace std;

class Test

{

private:

int x;

int y;

public:

Test(int x = 0, int y = 0) { this->x = x; this->y = y; }

Test &setX(int a) { x = a; return \*this; }

Test &setY(int b) { y = b; return \*this; }

void print() { cout << "x = " << x << " y = " << y << endl; }

};

int main()

{

Test obj1(5, 5);

// Chained function calls. All calls modify the same object

// as the same object is returned by reference

obj1.setX(10).setY(20);

obj1.print();

return 0;

}

|  |
| --- |
| Question 3  Wrong |

Predict the output of following C++ program.

|  |  |
| --- | --- |
| #include<iostream>  using namespace std;    class Test  {  private:    int x;  public:    Test(int x = 0) { this->x = x; }    void change(Test \*t) { this = t; }    void print() { cout << "x = " << x << endl; }  };    int main()  {    Test obj(5);    Test \*ptr = new Test (10);    obj.change(ptr);    obj.print();    return 0;  } | |
| A | x = 5 |
| B | x = 10 |
| C | Compiler Error |
| D | Runtime Error |

Explanation:

**this is a const pointer**, so there is an error in line "this = t;"

|  |
| --- |
| Question 4 |

Predict the output of following C++ program

|  |  |
| --- | --- |
| #include<iostream>  using namespace std;    class Test  {  private:    int x;    int y;  public:    Test(int x = 0, int y = 0) { this->x = x; this->y = y; }    static void fun1() { cout << "Inside fun1()"; }    static void fun2() { cout << "Inside fun2()"; this->fun1(); }  };    int main()  {    Test obj;    obj.fun2();    return 0;  } | |
| A | Inside fun2() Inside fun1() |
| B | Inside fun2() |
| C | Inside fun1() Inside fun2() |
| D | Compiler Error |

Explanation:

There is error in fun2(). It is a static function and tries to access this pointer. this pointer is not available to static member functions as static member function can be called without any object.

**“delete this” in C++**

Ideally *delete* operator should not be used for *this* pointer. However, if used, then following points must be considered.

1) *delete* operator works only for objects allocated using operator *new* (See [this post](https://www.geeksforgeeks.org/g-fact-30/)). If the object is created using new, then we can do *delete this*, otherwise behavior is undefined.

|  |
| --- |
| class A  {    public:      void fun()      {          delete this;      }  };    int main()  {    /\* Following is Valid \*/    A \*ptr = new A;    ptr->fun();    ptr = NULL; // make ptr NULL to make sure that things are not accessed using ptr.        /\* And following is Invalid: Undefined Behavior \*/    A a;    a.fun();      getchar();    return 0;  } |

2) Once *delete this* is done, any member of the deleted object should not be accessed after deletion.

|  |
| --- |
| #include<iostream>  using namespace std;    class A  {    int x;    public:      A() { x = 0;}      void fun() {        delete this;          /\* Invalid: Undefined Behavior \*/        cout<<x;      }  }; |

**The best thing is to not do *delete this* at all.**