**Rules:**

1. All the variable names or function names used in the program should be meaningful.

(Follow coding guidelines provided for reference)

1. For all the questions, provide the solutions based on your understanding of the concepts and provide necessary explanation to justify your answers (Do not execute and generate the output)
2. All the answers for this set shall be submitted in ***GitHub*** as a ***single file***.

**Questions:**

1. What happens if the base and derived class contains definition of a function with same prototype?
2. Compiler reports an error on compilation.
3. Only base class function will get called irrespective of object.
4. Only derived class function will get called irrespective of object.
5. Base class object will call base class function and derived class object will call derived class function.

ANSWER:

The function of the classes will be called based on the object creation. So option **D** is correct. Base class object will call base class function and derived class object will call derived class function.

1. What is the output of the below code snippet. Provide necessary explanation to support your view.

#include <iostream>

class BaseClass

{

int x;

public:

int y;

void set(int a,int b)

{

x=a;

y=b;

}

void show()

{

std::cout<<"X ="<<x;

}

};

class DerivedClass : public BaseClass

{

int i;

};

int main()

{

int baseSize,derivedSize;

BaseClass bc;

DerivedClass dc;

derivedSize=sizeof(dc);

baseSize=sizeof(bc);

std::cout<<"Size of BaseClass "<<baseSize << "\n";

std::cout<<"Size of DerivedClass "<<derivedSize << "\n";

return 0;

}

ANSWER:

Output:

Size of BaseClass 8

Size of DerivedClass 12

The sizeof(bc) function will get the maximum size allocated to each variable in that class. The base class has 2 int variable. The size of int is 4 so 4\*2 is 8. The derived class will inherit all the properties of the base class so it also inherits the variable of the base class. 2 int variable in base class + 1 int variable in derived class totally 3 int variables 3\*4 is 12.

1. In C++, why is it possible to make a base-class pointer point to a derived-class object
2. Because a derived-class object has all the properties (data and functions) of a base-class object, so it can be treated just like one for all intents and purposes (including being pointed at)
3. Because the compiler can automatically convert any base-class object to any required derived class
4. Because pointers in C++ aren't type-checked
5. All of the above
6. None of the above

ANSWER:

Because a derived-class object has all the properties (data and functions) of a base-class object, so it can be treated just like one for all intents and purposes (including being pointed at)

1. As an exception propagates it:
2. destructs any locally-declared objects created in any function through which it passes.
3. destructs any dynamically-allocated objects created in any function through which it passes.
4. destructs any locally-declared or dynamically-allocated objects created in any function through which it passes.
5. does not destruct any objects created in any function through which it passes.
6. only destructs objects declared in the try block which eventually catches it.

ANSWER:

Destructs any locally-declared or dynamically-allocated objects created in any function through which it passes.

1. What will be the output of the below code. Provide necessary information to support your decision.

#include <iostream>

class Printer {

public:

Printer() {}

~ Printer() {

throw 42;

}

};

int main(int argc, const char \* argv[]) {

try {

Printer hp;

throw 32;

} catch(int a) {

std::cout << a;

}

}

ANSWER:

Output:

Abort function will be called.

The process of calling destructors for automatic objects constructed on the path from a try block to a throw-expression is called “stack unwinding.” If a destructor called during stack unwinding exits with an exception, std::terminate is called

To avoid abort function **throw 42;** should be removed from the destructor.

1. Brief the difference between Class and Struct. Also, provide some information on when to use them.

ANSWER:

C++ origin’s in and compatibility with C.

C has struct. Struct has no concept of encapsulation. So that the members were public by default.

C++ has class. Class has concept of encapsulation. So that the members were private by default.

Changing semantics of struct so that its member can be private by default would broke the compatibility. So new keyword has been introduced in C++ that was **class**. So it was first named as “C with classes”.

1. What is wrong in this program? If there is any problem with the below code, provide the corrected code.

class Human

{

public:

Human() {}

~ Human(){}

};

class Pilot: public Human

{

public:

Pilot():Human(){}

~ Pilot(){}

};

int main(void)

{

Human\* george = new Pilot();

delete george;

}

ANSWER:

1. Valid username will:

[1] have a length between 6 to 9 [6 and 9 inclusive]

[2] contain only upper and/or lowercase alphabets

Complete the method **checkUsername** so that it validates the **username**.

* If the **username** length is less than 2, throw "**BadLengthException**"
* For other cases, "**checkUsername**" method shall return **true** or **false**.

**NOTE:**

1. You shall add your code only in the places where *"//Your code, if any"* is available.

2. Do not remove/replace/move other statement.

|  |
| --- |
| bool checkUsername(String username)  {  *//Your code, if any*  }  int main()  {  string username;  cin ≫ username;  *//your code, if any*  bool isValid = checkUsername(username);  if(isValid)  {  count≪ "Valid"  }  else  {  cout≪ "Invalid";  }  *//your code, if any*  } |

ANSWER:

bool checkUsername(String username)

{

**int namelength = username.length();**

**if (namelength < 2) {**

**throw string("Bad Length Exception");**

**}**

**else {**

**if ( namelength > 5 && namelength < 10) {**

**return true;**

**}**

**else {**

**return false;**

**}**

**}**

}

int main()

{

string username;

cin ≫ username;

**try {**

bool isValid = checkUsername(username);

if(isValid)

{

count≪ "Valid"

}

else

{

cout≪ "Invalid";

}

**}**

**catch (string excep) {**

**cout << excep;**

**}**

}

1. Why does this example fail? When run it prints out:

Copy constructor called

Copy constructor called

over and over.

1 #include <iostream.h>

2

3 class trouble {

4 public:

5 int data;

6

7 trouble(void);

8 trouble(const trouble &old);

9 trouble operator = (trouble old\_trouble);

10 };

11

12 trouble::trouble(void) {

13 data = 0;

14 }

15

16 trouble::trouble(const trouble &old) {

17 cout << "Copy constructor called\n";

18 \*this = old;

19 }

20

21 trouble trouble::operator = (trouble old\_trouble) {

22 cout << "Operator = called\n";

23 data = old\_trouble.data;

24 return (\*this);

25 }

26

27 int main()

28 {

29 trouble troublel;

30 trouble trouble2(troublel);

31

32 return (0);

33 }

ANSWER:

The above program fails because the parameter to this function

trouble trouble::operator = (trouble old\_trouble) {

is being passed as call-by-value parameter. When C++ see this type of parameter it calls the copy constructor to put the parameter in its stack.

So we have infinite “copy constructor” loop.

Solution to this problem is to pass parameter as reference to the function. We should reference the class instead of copy of the class.

trouble& trouble::operator = (const trouble& old\_trouble) {

1. Why does this example fail when we delete the variable list\_ptr? The

program seems to get upset when it tries to call clear at line 20.

1 #include <iostream.h>

2 #include <stdlib.h>

3

4 class list {

5 private:

6 int item; // Current item number

7

8 public:

9 virtual void clear() = 0;

10

11 void next\_item(void) {

12 ++item;

13 }

14

15 list(void) {

16 item = 0;

17 }

18

19 virtual ~list() {

20 clear();

21 }

22 };

23

24 class list\_of\_integers : public list {

25 public:

26 int array[100]; // Place to store the items

27

28 void clear(void) {

29 int i; // Array index

30

31 for (i = 0; i < 100; ++i)

32 array[i] = 0;

33 }

34 };

35

36 main()

37 {

38 list\_of\_integers \*list\_ptr = new list\_of\_integers;

39

40 // Cause problems

41 delete list\_ptr;

42 list\_ptr = NULL;

43 return (0);

44 }

ANSWER:

1. Why does this example work on UNIX, but when we run it in

MSDOS/Windows we get the message:

oot

ew able: file not found

#ifndef \_\_MSDOS\_

#define NAME "/root/new/table"

#else \_\_MSDOS\_

#define NAME "\root\new\table"

#endif \_\_MSDOS\_

in\_file.open(NAME);

if (in\_file.bad) {

cout << NAME << ": file not found\n";

exit (8);

}

ANSWER:

The problem is that the C uses the backslash (\) as escape character. So the character \r is a carriage return, \n is newline and \t is tab. What the compiler will think it as

<return>oot<newline>ew<tab>able

The string value should be specified by using double backslashes (\\) instead of single backslash (\).

“\\root\\new\\table”

If we use #include which uses filename, not a C string. Then we can specify the filename as

“\root\new\table” which is supported.

1. Why does this example not print an error message when an incorrect

command is entered?

#include <iostream.h>

#include <stdlib.h>

main()

{

char line[10];

while (1) {

cout << "Enter add(a), delete(d), quit(q): ";

cin.getline(line, sizeof(line));

switch (line[0]) {

case 'a':

cout << "Add\n";

break;

case 'd':

cout << "Delete\n";

break;

case 'q':

cout << "Quit\n";

exit(0);

defualt:

cout << Error: Bad command << line[0] << '\n';

break;

}

}

return (0);

}

ANSWER:

The compiler didn’t see our default line because we misspelled **default** as **defualt** which is a valid goto label. That why the error message was not displayed when we entered incorrect command.