Polytechnic Tutoring Center

Exam 2 Review - CS 2124, Spring 2026

Disclaimer: This mock exam is only for practice. It was made by tutors in the Polytechnic Tutoring Center and is not representative of the actual exam given by the CS Department.

1. What will be the result of running this code?

a. Compilation error

b. Runtime error

c. Undefined behavior

d. A: 21 B: 17

e. A: 19 B: 15

f. A: 21 B: 15

g. A: 17 B: 13

h. A: 21 B: 19

```
int main(){
int SIZE =12;
int* arr = new int[SIZE];
for(int i = 0; i<SIZE; i++){
arr[i] = 2*i -1;
}
int* ptr1 = arr + SIZE - 2;
int* ptr2 = ptr1 - 1;
std::cout<<"A: "<<ptr1[1]<<'\t';
std::cout<<"B: "<<*ptr2<<std::endl;
return 0;
}
```

2. What is the result of running this code?

a. Compilation error
b. 17
c. "Some Number"
d. Runtime error

```
class Foo {
public:
    Foo() {}
    void func(int x) { cout << x << endl; }
};

class Bar : public Foo {
public:
    Bar() {}
    void func() { cout << "Some Number" << endl; }
};

int main() {
    Bar b;
    b.func(17);
}
```

3. What will be the result of running the code below?

```
class Veggie{
public:
    virtual void display() const = 0;
    void eat(const Veggie& veg) const { cout << "In my crazy world, it's dog eat dog and veggie eat
veggie!" << endl; }
};

class Carrot : public Veggie{
public:
    void display() const override { cout << "Carrot is a veggie." << endl; }
    void eat(const Veggie& veg) const { cout << "In my crazy world, carrot preys on veggies!" << endl;
}
    void eat(const Carrot& veg) const { cout << "In my crazy world, veggie preys on carrot!" << endl; }
};

class Pumpkin : public Carrot{
public:
    virtual void display() const override { cout << "In my crazy world, pumpkin is a carrot." << endl;
}
    void eat(const Veggie& veg) const { cout << "If you only eat veggie, you won't feel full; unless
it's a pumpkin!" << endl; }
};

int main(){

    Pumpkin Sam;
    Carrot* cp = &Sam;
    Veggie* vp = cp;
    Carrot Car = Sam; #Line A

    cp->display();
    vp->display();
    Car.display();
    Sam.eat(*vp);
    Car.eat(*cp); # Line B

}
```

a.
In my crazy world, pumpkin is a carrot.
In my crazy world, pumpkin is a carrot.
Carrot is a veggie.
If you only eat veggie, you won't feel full; unless it's a pumpkin!
In my crazy world, veggie preys on carrot!

b.
In my crazy world, pumpkin is a carrot.
In my crazy world, pumpkin is a carrot.
Carrot is a veggie.
If you only eat veggie, you won't feel full; unless it's a pumpkin!
In my crazy world, carrot preys on veggies!

c.
Carrot is a veggie.
Carrot is a veggie.
Carrot is a veggie.
If you only eat veggie, you won't feel full; unless it's a pumpkin!
In my crazy world, veggie preys on carrot!

d.
Compilation error at Line A

e.
Compilation error at Line B

f.
None of the above

4. What will be the output of the following code?

```
class A {
public:
    A() { cout << "A()\n"; }
    ~A() { cout << "~A()\n"; }
};

class B : public A {
public:
    B() { cout << "B()\n"; }
    ~B() { cout << "~B()\n"; }
};

class C : public A {
public:
    C() { cout << "C()\n"; }
    ~C() { cout << "~C()\n"; }
};

int main() {
    C c;
    cout << "Finished!" << endl;
}
```

a.

```
A()
C()
~C()
~A()
Finished!
```

b.

```
A()
C()
Finished!
~C()
~A()
```

c.

```
C()
B()
A()
~C()
~B()
~A()
Finished!
```

d.

```
A()
C()
Finished!
~A()
~C()
```

5. What is the result of the following code?

a. Compilation error at line A

b. Compilation error at line B

c. Compilation error at line C

d. Compilation error at line D

e. Runtime error

f. Undefined Behaviour

g. None of the above

```
class Guitar {
private:
    string name;
public:
    Guitar(string& aString) : name(aString) {}
    Guitar() : name() {}
    virtual void getName() const {
        cout << name << endl;
    }
};

class ElectricGuitar : public Guitar {
public:
    ElectricGuitar(string& aString) :
Guitar(aString) {}
};

class AcousticGuitar : public Guitar {

};

int main(){
    Guitar a;// line A
    ElectricGuitar b("Electric Guitar"); // line B
    Guitar* ap = &a; // line C
    Guitar* bp = &b; // line D
```

```
}
```

6. Given the same class definitions in question 5, along with the below function definitions, what is the output of the program?

```
void func(Guitar& a) { cout << "foo(A)\n"; };
void func(ElectricGuitar& b) { cout << "foo(B)\n"; };
void func(AcousticGuitar& c) { cout << "foo(C)\n"; };

int main(){
    Guitar a;
    ElectricGuitar b;
    AcousticGuitar c;
    func(a);
    func(b);
    func(c);
}
```

a.	b.	c.	d.
foo(A) foo(A) foo(A)	foo(A) foo(B) foo(A)	Compilation error	foo(A) foo(B) foo(B)

7. What is the output of the following code

a. 4 8 10 6 6 8
b. 4 8 10 8 8 10
c. 4 8 10 4 6 8
d. 2 6 8 4 6 8
e. None of the above

```
int main() {
    int data[5] = { 2, 4, 6, 8, 10 };
    int* p = data + 1;
    cout << *p << " ";
    cout << *(p + 2) << " ";
    cout << p[3] << " ";
    cout << *(++p) << " ";
    cout << *p++ << " ";
    cout << *p << endl;
}
```

8. What is the result of the following code?

a. I'm a Circle

I'm a Triangle

b. Runtime error due to function overriding

c. Runtime error due to function overloading

```
class Shape {
public:
    Shape() {}
    virtual void draw() = 0;
};

class Circle : public Shape {
public:
    Circle() {}
    void draw() { cout << "I'm a Circle" << endl; }
};
class Triangle : public Shape {
public:
```

d. Compilation error due to function overloading

e. Compilation error due to function overriding

f. None of the above

```
    Triangle() {}
    void display() { cout << "I'm a Triangle" <<
endl; }
};

int main() {
    Circle c;
    Triangle t;
    c.draw();
    t.display();
}
```

9. Given the following class definition for `Thing`, write the `operator=` function for that class.

```
class Thing {
public:
    Thing(int x) { p = new int(x); }
    Thing(const Thing& anotherThing) {
        p = new int(*anotherThing.p);
    }
    ~Thing() {
        delete p;
        p = nullptr;
    }
private:
    int* p; };
```

10. Implement two classes:

The base class, which should be named *PetProject*, has **only one member variable**, an integer *price* stored on the heap, because we can have a lot of pet projects and they all are valuable and worth a price!

The derived class, *MasterProject*, also has **only one member variable**, an integer *multiplier* stored on the heap. Now here is the trick: When a project is so good, we call it a master project. Every master project is of course first our pet project, but not all pet projects are good enough to be master projects! And of course a master project is worth a lot more! You need to implement the following functionalities:

- output operator
  - which simply outputs the networth of a project, for PetProject, it is simply its price member
  - for a MasterProject, it is the product of its price, and the multiplier.
- Copy control
- bool operator
  - which returns true when the networth of the project is non-zero (negative is fine. Some projects are like that.), and false when the networth is zero
  - And keep in mind some nonsense code such like this should crash:
- `cout<<myProject + 1; /**myProject is an instance of a MasterProject**/`
- equality operator, which compares if two projects have the same net worth

An example of your code behavior is such:

```
int main(){
    PetProject Pet(3);
    std::cout<<"Pet: "<<Pet<<std::endl;

    MasterProject Master(3,8);
    std::cout<<"Master: "<<Master<<std::endl;
    std::cout<<(Master?"Not Zero" : "Zero")<<std::endl;

    return 0;
}
```

The output is:

```
Pet: 3
Master: 24
Not Zero
Process finished with exit code 0
```

11. Write two classes, `Employer` and `Employee` or `Boss` and `Employee` (whatever floats your boat), that have the following characteristics…

- An Employer, has a set of underlings which he may boss around
- He should know who his Employees are
- All Employees should know who their Employer is if they have one
- Any Employee should be able to quit at any given time
- Any Employer should be able to fire any of their Employees at any given time
- An Employer should also be able to hire an Employee so long as he is not already employed
- In the event that an Employer hires an Employee, that Employer becomes the care taker of that particular Employee, if the Employer loses that Employee, things could get messy
- These quitting and hire functions should not fail silently
- All Employees are created on the heap
- Implement the big three for Employers
- When an Employee is first born, he does not have a boss, but he is always born with a name
- When an Employer is first born, he does not have any Employees but is always born with a name unless he is copied from another Employer
- The main idea here is that the two classes sort of know about each other, find a way to make this possible
- Note that Employees and Employers can have the same name but still be different. Thus, for a company to be able to differentiate between Employees, the name should not be the distinguishing factor!
- The fact that Employees may be different despite having the same name, suggests that it may be more convenient for the Employer to store its Employees in a vector of Employee pointers. In fact, this isn't a suggestion, it is a must.

0. (Extra Credit) Who created Java?

a. James Gosling
a. Ada Lovelace
a. Alan Turing
a. Dennis Ritchie
a. Claude Shannon
a. Bjarne Stroustrup
a. none of the above