CSE 333 – Section 5: C++ Intro

Const & References

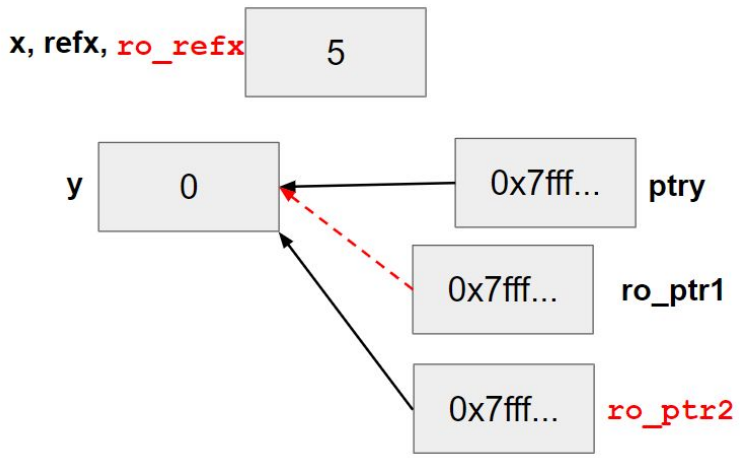
1) Consider the following functions and variable declarations - Also covered in lecture slides.

a) Draw a memory diagram for the variables declared in `main`.

```
void foo(const int &arg);
void bar(int &arg);
```

```
int main(int argc, char **argv)
{
  int x = 5;
  int &refx = x;
  const int &ro_refx = refx;
  int y = 0;
  int *ptry = &y;
  const int *ro_ptr1 = &y;
  int *const ro_ptr2 = &y;

  // ...
}
```



b) When would you prefer `void func(int &arg);` to `void func(int *arg);`? Expand on this distinction for other types besides `int`.

- When you don't want to deal with pointer semantics, use references
- When you don't want to copy stuff over (doesn't create a copy, especially for parameters and/or return values), use references
- Style wise, we want to use **references for input parameters** and **pointers for output parameters**, with the output parameters declared last

c) What does the compiler think about the following lines of code:

```
bar(refx);                  // No issues
bar(ro_refx);               // Compiler error - ro_refx is const
foo(refx);                  // No issues
```

d) How about this code?

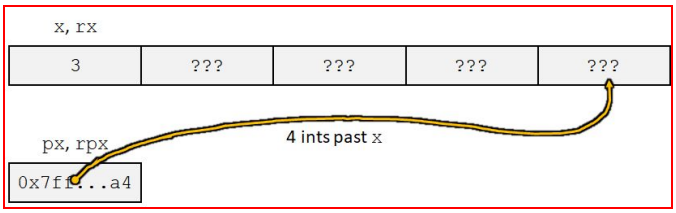
```
ro_ptr1 = (int*) 0xDEADBEEF;    // No issues
ro_ptr2 = ro_ptr2 + 2;          // Compiler error - ro_ptr2 is const
*ro_ptr1 = *ro_ptr1 + 1;        // Compiler error - (*ro_ptr1) is const
```

2) What does the following program print out? (5 min) Hint: box-and-arrow diagram!

```
int main(int argc, char** argv) {
  int x = 1;       // assume &x = 0x7ff...94
  int& rx = x;
  int* px = &x;
  int*& rpx = px;

    rx = 2;
  *rpx = 3;
   px += 4;

  cout << "  x: " <<   x << endl;  //   x: 3
  cout << " rx: " <<  rx << endl;  //  rx: 3
  cout << "*px: " << *px << endl;  // *px: ??? (garbage)
  cout << " &x: " <<  &x << endl;  //  &x: 0x7ff...94
  cout << "rpx: " << rpx << endl;  // rpx: 0x7ff...a4
  cout << "*rpx: " << *rpx << endl; // *rpx = *px: ??? (garbage)
  return 0;
}
```



3) Refer to the following poorly-written class declaration. (10 min)

```
class MultChoice {
 public:
  MultChoice(int q, char resp) : q_(q), resp_(resp) { }  // 2-arg ctor
  int get_q() const { return q_; }
  char get_resp() { return resp_; }
  bool Compare(MultChoice &mc) const;  // do these MultChoice's match?

 private:
  int  q_;     // question number
  char resp_;  // response: 'A','B','C','D', or 'E'
};  // class MultChoice
```

a) Indicate (**Y**/**N**) which *lines* of the snippets of code below (if any) would cause compiler errors:

Code Snippets	Error?

`int z = 5;`
`const int *x = &z;`
`int *y = &z;`
`x = y;`
`*x = *y;`
	N
	N
	N
	N
	Y

`const MultChoice m1(1,'A');`
`MultChoice m2(2,'B');`
`cout << m1.get_resp();`
`cout << m2.get_q();`
	N
	N
	Y
	N

Code Snippets	Error?

`int z = 5;`
`int *const w = &z;`
`const int *const v = &z;`
`*v = *w;`
`*w = *v;`
	N
	N
	N
	Y
	N

`const MultChoice m1(1,'A');`
`MultChoice m2(2,'B');`
`m1.Compare(m2);`
`m2.Compare(m1);`
	N
	N
	N
	Y

b) What would you change about the class declaration to make it better? Feel free to mark directly on the class declaration above if desired. **(optional)**
Many possibilities. Importantly, make get_resp() const and make parameter to Compare() const. Stylistically, probably makes sense to add a setter method and default constructor. Could also optionally disable copy constructor and assignment operator.

4) Mystery Functions (10 min)

Consider the following C++ code, which has **__???__** in the place of 3 function names in `main`:

```
struct Thing {
  int a;
  bool b;
};

void PrintThing(const Thing& t) {
  cout << boolalpha << "Thing:  " << t.a << ", " << t.b << endl;
}

int main() {
  Thing foo = {5, true};
  cout << "(0) ";
  PrintThing(foo);

  cout << "(1) ";
  __???__(foo);   // mystery 1:  f2
  PrintThing(foo);

  cout << "(2) ";
  __???__(&foo);  // mystery 2:  f3
  PrintThing(foo);

  cout << "(3) ";
  __???__(foo);   // mystery 3:  f1, f2, f4, or f5
  PrintThing(foo);

  return 0;
}
```

Program Output:	**Possible Functions:**
(0) Thing: 5, true (1) Thing: 6, false (2) Thing: 3, true (3) Thing: 3, true	void **f1**(Thing t); void **f2**(Thing &t); void **f3**(Thing *t); void **f4**(const Thing &t); void **f5**(const Thing t);

List *all* of the possible functions (**f1** - **f5**) that could have been called at each of the three mystery points in the program that would compile cleanly (no errors) and could have produced the results shown. There is at least one possibility at each point; there might be more.

- Hint: look at parameter lists and types in the function declarations and in the calls.

5) Building a Rectangle

Define a class Rectangle that has two member variables that are two Point Objects. These two points will be used to represent the upper left and lower right corners of the rectangle. The declaration of the point class is given below:

```
class Point {
 public:
  // Constructs a point that represents the Cartesian point (x, y)
  Point(int x, int y);

  // Returns the x component of this point.
  int get_x() { return x_; }

  // Returns the y component of this point.
  int get_y() { return y_; }

  // Returns the distance between this point and the provided point.
  double Distance(Point & p);

  // Sets the current point to represent the passed in coordinates.
  void SetLocation(int x, int y);

 private:
  int x_, y_;
};
```

Be sure that you follow good c++ practices, and use const appropriately. Your rectangle class should contain the following methods:

- at least one constructor
- **getul()** and **getlr()** - getters that return the upper-left and lower-right corners of the rectangle
- **area()** - returns the Rectangle's area.
- **contains(Point &p)** - returns true iff the point is inside the rectangle, and false otherwise.

Rectangle.h:

```
#ifndef RECTANGLE_H_
#define RECTANGLE_H_

#include "./Point.h"

class Rectangle {
 public:
  Rectangle(const Point &ul, const Point &lr);
  Point getul() const
  Point getlr() const
  int area() const;
```

```
  bool contains(const Point &p) const;
 private:
  Point ul_, lr_;
};

#endif  // RECTANGLE_H_
```

Rectangle.cc:

```
#include "./Rectangle.h"

Rectangle::Rectangle(const Point &ul, const Point &lr) :
  ul_(ul.get_x(), ul.get_y()),
  lr_(lr.get_x(), lr.get_y()) { }

Point Rectangle::getul() const {
  return this->ul_;
}

Point Rectangle::getlr() const {
  return this->lr_;
}

int Rectangle::area() const {
  int horizontal = this->lr_.get_x() - this->ul_.get_x();
  int vertical = this->ul_.get_y() - this->lr_.get_y();
  return horizontal * vertical;
}

bool Rectangle::contains(const Point &p) const {
  bool x_in = (p.get_x() <= lr_.get_x()) && (p.get_x() >= ul_.get_x());
  bool y_in = (p.get_y() <= ul_.get_y()) && (p.get_y() >= lr_.get_y());
  return x_in && y_in;
}
```