# CSE 333 – Section 5: C++ Intro

Welcome back to section! We're glad that you're here :)

## *References*

References create *aliases* that we can bind to existing variables. References are not separate variables and cannot be reassigned after they are initialized. In C++, you define a reference using: **`type &name = var`**. The '`&`' is similar to the '`*`' in a pointer definition in that it modifies the type and the space can come before or after it.

## *Const*

Const makes a variable *unchangeable* after initialization, and is enforced at compile time.

```
const int x = 5;                // Can't assign to x
const int* xptr = &x;           // Can assign to xptr, but not *xptr
int *const yptr = &y;           // Can assign to *yptr, but not yptr
const int *const zptr = &z;     // Can't assign to *zptr or zptr
```

Class objects can be declared const too - a const class object can only call member functions that have been declared as const, which are not allowed to modify the object instance it is being called on.

Exercises:

**1) Consider the following functions and variable declarations.**

a) Draw a memory diagram for the variables declared in `main`. It might be helpful to distinguish variables that are constant in your memory diagram.

```
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`.

c) If we have functions void foo(const int &arg); and void bar(int &arg);, what does the compiler think about the following lines of code:

```
bar(refx);
bar(ro_refx);
foo(refx);
```

d) How about this code?

```
ro_ptr1 = (int*)0xDEADBEEF;
ro_ptr2 = ro_ptr2 + 2;
*ro_ptr1 = *ro_ptr1 + 1;
```

**2) Extra practice: What does the following program print out?** 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;
  cout << " rx: "  <<  rx  << endl;
  cout << "*px: "  << *px  << endl;
  cout << " &x: "  <<  &x  << endl;
  cout << "rpx: "  << rpx  << endl;
  cout << "*rpx: " << *rpx << endl;

  return EXIT_SUCCESS;
}
```

**3) Refer to the following *poorly-written* class declaration.**

```
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;`<br>`const int *x = &z;`<br>`int *y = &z;`<br>`x = y;`<br>`*x = *y;` | |
| `const MultChoice m1(1,'A');`<br>`MultChoice m2(2,'B');`<br>`cout << m1.get_resp();`<br>`cout << m2.get_q();` | |

| **Code Snippets** | **Error?** |
|---|---|
| `int z = 5;`<br>`int *const w = &z;`<br>`const int *const v = &z;`<br>`*v = *w;`<br>`*w = *v;` | |
| `const MultChoice m1(1,'A');`<br>`MultChoice m2(2,'B');`<br>`m1.Compare(m2);`<br>`m2.Compare(m1);` | |

b) What would you change about the class declaration to make it better? Feel free to mark directly on the class declaration above.

**4) Mystery Functions** <u>**(EXTRA PRACTICE)**</u>
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
  PrintThing(foo);

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

  cout << "(3) ";
  __???__(foo);   // mystery 3
  PrintThing(foo);

  return 0;
}
```

| **Program Output:** | **Possible Functions:** |
|---|---|
| `(0) Thing:  5, true`<br>`(1) Thing:  6, false`<br>`(2) Thing:  3, true`<br>`(3) Thing:  3, true` | `void f1(Thing t);`<br>`void f2(Thing &t);`<br>`void f3(Thing *t);`<br>`void f4(const Thing &t);`<br>`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.

- <u>Hint</u>: 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.