CSE 333 – Section 4: C++ Intro

**Const & References**

1) Consider the following functions and variable declarations

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

   ```cpp
   void foo(const int& arg);
   void bar(int& arg);
   int main(int argc, char **argv) {
       int x = 5;
       int& x_ref = x;
       int* x_ptr = &x;
       const int& ro_x_ref = x;
       const int* ro_ptr1 = &x;
       int* const ro_ptr2 = &x;
       // ...
   }
   ```

   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:

      ```cpp
      bar(x_ref);       // No issues
      bar(ro_x_ref);   // Error - ro_x_ref is const
      foo(x_ref);      // No issues
      ```

   d) How about this code?

      ```cpp
      ro_ptr1 = (int*) 0xDEADBEEF;       // No issues
      x_ptr = &ro_x_ref;                // Error - ro_x_ref is const
      ro_ptr2 = ro_ptr2 + 2;            // Error - ro_ptr2 is const
      *ro_ptr1 = *ro_ptr1 + 1;          // Error - (*ro_ptr1) is const
      ```

   e) In a function `const int f(const int a);` are the const declarations useful to the client? How about the programmer? What about this function needs to change to make const matter?

      The const return and parameter both don’t affect the client at all, since they work with copies of the parameter/return value. This enforces the programmer not to modify `a` at all. If `f` used references for the parameter/return, then it would matter to both the client and the programmer.
2) 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:

<table>
<thead>
<tr>
<th>Code Snippets</th>
<th>Error?</th>
</tr>
</thead>
<tbody>
<tr>
<td>const MultChoice m1(1,'A');</td>
<td>N</td>
</tr>
<tr>
<td>MultChoice m2(2,'B');</td>
<td>N</td>
</tr>
<tr>
<td>cout &lt;&lt; m1.getResp();</td>
<td>N</td>
</tr>
<tr>
<td>cout &lt;&lt; m2.getQ();</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code Snippets</th>
<th>Error?</th>
</tr>
</thead>
<tbody>
<tr>
<td>const MultChoice m1(1,'A');</td>
<td>N</td>
</tr>
<tr>
<td>MultChoice m2(2,'B');</td>
<td>N</td>
</tr>
<tr>
<td>m1.Compare(m2);</td>
<td>N</td>
</tr>
<tr>
<td>m2.Compare(m1);</td>
<td>Y</td>
</tr>
</tbody>
</table>

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 the parameter to Compare() const. Stylistically, it makes sense to add a setter method and default constructor. Could also optionally disable copy constructor and assignment operator.
3. Refer to the following file definitions.

```
Point.h    class Point { ... };
UsePoint.cc #include "Point.h"
             #include "Thing.h"
             int main( ... ) { ... }
UseThing.cc #include "Thing.h"
             int main( ... ) { ... }
Alone.cc    int main( ... ) { ... }
```

a. Draw out Point’s DAG
(The direction of the arrows is not important, but be consistent)

```
UsePoint.cc  Point.h  Point.cc  Thing.h  UseThing.cc  Alone.cc
             UsePoint.o  Point.o
             UsePoint    UseThing    Alone
             all
```

b. Write the corresponding Makefile for Point

```
CFLAGS = -Wall -g -std=c++17
all: UsePoint UseThing Alone
UsePoint: UsePoint.o Point.o
          g++ $(CFLAGS) -o UsePoint UsePoint.o Point.o
UsePoint.o: UsePoint.cc Point.h Thing.h
           g++ $(CFLAGS) -c UsePoint.cc
Point.o: Point.cc Point.h
         g++ $(CFLAGS) -c Point.cc
UseThing: UseThing.cc Thing.h
         g++ $(CFLAGS) -o UseThing UseThing.cc
Alone: Alone.cc
       g++ $(CFLAGS) -o Alone Alone.cc
clean:
         rm UsePoint UseThing Alone *.o *~
```
Bonus: Const Const Const
Which of the following lines will result in a compiler error?

<table>
<thead>
<tr>
<th>Code Snippets</th>
<th>Error?</th>
</tr>
</thead>
</table>
| int z = 5; \  
  const int* x = &z; \  
  int* y = &z; \  
  x = y; \  
  *x = *y; | N       |

<table>
<thead>
<tr>
<th>Code Snippets</th>
<th>Error?</th>
</tr>
</thead>
</table>
| int z = 5; \  
  int* const w = &z; \  
  const int* const v = &z; \  
  *v = *w; \  
  *w = *v; | N       |

Bonus: What does the following program print out?  Hint: box-and-arrow diagram!

```cpp
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;
}
```
**Bonus: Mystery Functions**

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

```cpp
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;
}
```

Possible Functions:

- `void f1(Thing t);`
- `void f2(Thing& t);`
- `void f3(Thing* t);`
- `void f4(const Thing& t);`
- `void f5(const Thing t);`

**Program Output:**

(0) Thing: 5, true  
(1) Thing: 6, false  
(2) Thing: 3, true  
(3) Thing: 3, true

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.