Advanced Programming Concepts with C++

CSI 2372

Tutorial # 3

Selected exercises from a previous midterm exam and chapter 6

uOttawa

Ahmedou Jreivine

PART A: SHORT QUESTIONS

1. Clearly mark any lines causing a compile error below [1]

```
const int ci = 2;
int j = ci;
ci = j;
int i = 5;
const int cj = i;
std::cin >> ci;
```

2. Call the function getAddIncrementCount of struct A and print the return value to console, what will be printed? [1]

```
struct A {
    static int count;
    static int getAddIncrementCount() {
        return ++count;
    }
};
int A::count{0};
int main() {
  cout << A:: getAddIncrementCount()<<endl;    // it will return 1
}
```

PART A: SHORT QUESTIONS (A 2016)

1. Given the following declaration
   - void arrays(int (*a)[3], int (&b)[3]);
   - call the function arrays with the arguments: int argA[3], argB[3];

- **Solution:**

```
// Call the following function
void arrays( int (*a)[3], int (&b)[3] ) {
}

void passArray() {
        int argA[3], argB[3];
        arrays( &argA, argB );
}
```

PART A: SHORT QUESTIONS (A 2016)

2. What is printed by the following function

```
void printArray() {
  int array[][2]{1,2,3,4,5,6};
  cout << (*(array+1))[2] << endl;
  return;
}
// 5 is printed
```

3. What is printed by the following?

```
bool bv = true;
short sv = 2;
int iv = 1;
cout << (iv < sv && bv) << endl;
//1 is printed
```

PART A: SHORT QUESTIONS (A 2016)

4. What is printed by the following?

```
unsigned int ua = 4, ub = 2;
cout << (ua ^ ub | 1) << endl;
```

// **7** is printed

5. What is printed by the following?

```
char cA[]{"Hello World"};
*(cA+5) = 0;
cout << cA << endl;
```

// ***Hello*** is printeed

PART A: SHORT QUESTIONS (A 2016)

6. What is printed by the following?

```
char abc[]{"abc"};
for ( auto v : abc ) {
    v++;
}
cout << abc << endl;
 // abc is printed
for ( auto& v : abc ) {
    v++;
}
abc[3] = 0;
cout << abc << endl;
 // bcd is printed
```

PART A: SHORT QUESTIONS (A 2016)

7. What is printed by the following?

```
        int i=7,j=2;
        auto k = i/j;
        auto m = i%j;
        cout << k << " and " << m << endl;
    // 3 and 1 is printed.
```

8. What is printed by the following ?

```
        int i = 2;
        int& j = i;
        auto k = j;
        decltype(j) m = j;
        --i;
        cout << k << endl;
        cout << m << endl;
        // 2 and 1 are printed
```

PART A: SHORT QUESTIONS (A 2017)

```
// What is printed by the following?
        std::string s;
        s+="1 ";
        s+='2';
        s[1] = '0';
        cout << s.c_str() << endl;  // 102 is printed
// What is printed by the following?
        double dd = 3.0;
        int ii = 2;
        char cc = 1;
        double rr = dd/ii+cc;
        cout << rr << endl;          // 2.5 is printed
```

PART A: SHORT QUESTIONS (A 2017)

- What is printed by the following program?

```
int aa[]{2,4,6};
int *pA = &aa[0];
int **pB = &pA;
++pA;
cout << **pB << endl;       // 4 is printed
```

- Use auto in the following variable definition to define exactly the same types

```
 int x = 3;
 // int &i = x;
auto& i = x; ++x; cout << i << endl;
  // const int *j = &x;
  const auto j = &x; ++x; cout << *j << endl;
  // j++;
```

PART A: SHORT QUESTIONS (A 2017)

5. Rewrite or mark up the function prototypes in the class LetterGrade using const and references as much as possible but not more. [1]

```
class LetterGrade {
        string d_mark{"INC"};
    public:
        LetterGrade() = default;
        LetterGrade( string m ): d_mark(m) {}
        string get(){ return d_mark; }
        void set(string m){d_mark = m;}
        bool pass(){ if ( d_mark < "D" || d_mark == "D+") return true; }
};
```

PART A: SHORT QUESTIONS (A 2017)

5. Solution

```
class LetterGrade {
        string d_mark{"INC"};
public:
        LetterGrade() = default;
        LetterGrade( const string& m ) : d_mark(m) {}
        string get() const { return d_mark; }
        void set(const string& m) {d_mark = m;}
        bool pass() const { if ( d_mark < "D" || d_mark == "D+") return true;}
};
```

Exercise 6.6:

Explain the differences between a parameter, a local variable, and a local static variable. Give an example of a function in which each might be useful.

Solution 6.6:

Local variable: Variables defined inside a **block**;

parameter: **a local variable** declared inside the **function parameter list**

Local static variable: local static variable, object, is initialized before the first time execution passes through the object’s definition.

Local statics are destroyed only when the program terminates.

```
// example:
      size_t count_add(int n)     // n is a parameter.
      {
        static size_t ctr = 0;   // ctr is a static variable.
        ctr += n;
        return ctr;
      }
      int main()
      {
        for (size_t i = 0; i != 10; ++i)  // i is a local variable.
          cout << count_add(i) << endl;

        return 0;
      }
```

Exercise 6.7:

Write a function that returns 0 when it is first called and then generates numbers in sequence each time it is called again.

- **Solution:**

```
size_t generate()
{
    static size_t ctr = 0;
    return ctr++;
}
```

Exercise 6.10:

Using pointers, write a function to swap the values of two ints. Test it.

```
#include <iostream>
#include <string>
#include <stdexcept>
 void swap(int* lhs, int* rhs) {
        int tmp;
        tmp = *lhs;
        *lhs = *rhs;
        *rhs = tmp;
}
int main(){
        for (int lft, rht; std::cout << "Please Enter:\n", std::cin >> lft >> rht; ){
                swap(&lft, &rht);
                std::cout << lft << " " << rht << std::endl;
        }
        return 0;
}
```

Exercise 6.11:

Write and test your own version of ***reset*** that takes a reference.

```
#include <iostream>
void resetInt(int &i) {
    i = 0;
}
int main() {
    int a;
    std::cin >> a;
    std::cout << "before reset: " << a << std::endl;
    resetInt(a);
    std::cout << "after reset:  " << a << std::endl;
    return 0;
}
```

Exercise 6.15:

- Explain the rationale for the type of each of find_char’s parameters In particular, why is s a reference to const but occurs is a plain reference?
- Why are these parameters references, but the char parameter c is not? What would happen if we made s a plain reference?
- What if we made occurs a reference to const?

Solution 6.15:

The function prototype is

string::size_type find_char(const string &s, char c, string::size_type &occurs)

- ***s*** and ***occurs*** are both references to avoid copy.
- ***s*** is const because it isn't changed inside function and a string literal can be used here.
- ***occurs*** is plain reference because it is used to pass information (changed inside function).
- ***c*** is nonreference because copy a char is very cheap. It's fine to make it a const reference but not plain reference, because we don't want to accidentally change ***c*** inside function, and we may want to pass a char literal to the function.
- If occurs is made a reference to const, then we cannot get how many times the character c occurred in string s.

Exercise 6.16:

- The following function, although legal, is less useful than it might be. Identify and correct the limitation on this function:
  - ***bool is_empty(const string& s) { return s.empty(); }***

- **Solution:**
  - Since this function doesn't change the argument, "const" shoud be added before string& s, otherwise this function is misleading and can't be used with const string or in a const function.

Exercise 6.20:

- When should reference parameters be references to const?
- What happens if we make a parameter a plain reference when it could be a reference to const?

- **Solution:**
  - If the reference parameters will not be changed inside function, then they should be reference to const.
  - If we make a parameter a plain reference, then we can not pass:
    - a const object,
    - or a literal,
    - or an object that requires conversion to a plain reference parameter.

Exercise 6.21:

- Write a function that takes an int and a pointer to an int and returns the larger of the int value or the value to which the pointer points.
- What type should you use for the pointer?

```
#include <iostream>
using std::cout;
int larger_one(const int i, const int *const p){
    return (i > *p) ? i : *p;
}
int main(){
    int i = 6;
    cout << larger_one(7, &i);
    system("pause");
    return 0;
}
```

Exercise 6.24:

- Explain the behavior of the following function. If there are problems in the code, explain what they are and how you might fix them.

```
void print(const int ia[10]) {
     for (size_t i = 0; i != 10; ++i)
          cout << ia[i] << endl;
}
```

- **Solution:**
  - The function prototype is the same as void print(const int *ia), which means we can pass any pointer to int to the function, not only an array of ten ints. This will lead to an error.
  - We can change the parameter to a reference to array:
  - `void print(const int (&ia)[10]) { /* ... */ }`

Exercise 6.54:

- Write a declaration for a function that takes two int parameters and returns an int, and declare a vector whose elements have this function pointer type.

```
#include <vector>
#include <iostream>
int foo(int, int);
int bar(int, int);
int main() {
    std::vector<int(*)(int, int)> vf;
    vf.push_back(foo);
    vf.push_back(bar);
    vf[0](1, 2);
    vf[1](3, 4);
    for (const auto &e : vf)
    e(9, 9);
    return 0;
}
int foo(int a, int b) {
    std::cout << "Called foo(" << a << ", " << b << ")" << std::endl;
    return 0;
}
int bar(int a, int b) {
    std::cout << "Called bar(" << a << ", " << b << ")" << std::endl;
    return 0;
}
```

Exercise 6.55:

- Write four functions that add, subtract, multiply, and divide two int values. Store pointers to these values in your vector from the previous exercise.

- **Solution:**

  - int add(int a, int b) { return a + b; }
  - int subtract(int a, int b) { return a - b; }
  - int multiply(int a, int b) { return a * b; }
  - int divide(int a, int b) { return b != 0 ? a / b : 0; }

Exercise 6.56:

- Call each element in the vector and print their result.

- **Solution:**

```
int func(int a, int b);
std::vector<decltype(func) *> vec{ add, subtract, multiply, divide };
for (auto f : vec)
     std::cout << f(2, 2) << std::endl;
```

Refereces

Accreditation:

- This presentation is prepared/extracted from the following resources:
  - C++ Primer, Fifth Edition.
    Stanley B. Lippman Josée Lajoie Barbara E. Moo
  - https://github.com/jaege/Cpp-Primer-5th-Exercises
  - https://github.com/Mooophy/Cpp-Primer