CS 7B - Fall 2018 - Midterm 1 —Name:

Write all responses on separate paper. Do not use an electronic computer, only your "necktop."

1. Write the number of the definition on the right next to the term it defines.

(a) **class** 3

(b) **constructor** 4

(c) **container** 12

(d) **copy** 1

(e) **invariant** 10

(f) **overload** 7

(g) **reference** 6

(h) **pointer** 5

(i) **scope** 2

(j) **byte** 11

(k) **type** 9

(l) **move** 8

(1) An operation that makes two objects have values that compare equal..

(2) The region of program text (source code) in which a name can be referred to.

(3) A user-defined type that may contain data members, function members, and member types.

(4) An operation that initializes an object. Typically establishes an invariant and often acquires resources needed for an object to be used (which are then typically released by a destructor).

(5) (1) a value used to identify a typed object in memory; (2) a variable holding such a value.

(6) (1) a value describing the location of a typed value in memory; (2) a variable holding such a value.

(7) Define two functions or operators with the same name but different argument (operand) types.

(8) an operation that transfers a value from one object to another, leaving behind a value representing "empty."

(9) Something that defines a set of possible values and a set of operations for an object.

(10) Something that must be always true at a given point (or points) of a program; typically used to describe the state (set of values) of an object or the state of a loop before entry into the repeated statement.

(11) The basic unit of addressing in most computers.

(12) An object that holds elements (other objects).

2. Consider the following complete program:

```
1 #include<iostream>
  int main() {
3     char* pc = 0;
      char& rc = *pc;
5     std::cout<<pc;
      std::cout<<rc;
7 }
```

(a) What is the variable name declared on line 3 and what is its type?
ANS: `pc` is a pointer to a character,

(b) How is the variable on line 3 initialized?
ANS: It is initialized to the NULL pointer (`nullptr`).

(c) What is the variable name declared on line 4 and what is its type?
ANS: The variable named `rc` is defined to be a reference to (the address of) a variable of type `char` whose value is the address pointed to by `pc`.

(d) How is the variable on line 4 initialized?
ANS: It is assigned the value of the address of the `char` pointed to by `pc`, which is then an alias for the `char` that `pc` purportedly points to...except `pc` is `NULL`. Bad!

(e) This code will likely compile but then crash on execution. Why?
ANS: Line 4 binds the reference `rc` to `nllptr`. As the C++11 ISO states, `[dcl.ref] [...] a null reference cannot exist in a well-defined program, because the only way to create such a reference would be to bind it to the "object" obtained by dereferencing a null pointer, which causes undefined behavior.`

3. Consider the following complete program:

```
1  #include <iostream>
   using std::cout;
3  #include <fstream>
   using std::ifstream;
5  #include <string>
   using std::string;
7  //precondition: a string and a reference to an input file stream
   //postcondition: true only if string is in the file after the point referenced by the ifstream
9  bool isWord(const string& w, ifstream& fs) {
     string word;
11   while (fs >> word) {
       if (word == w) return true;
13   }
     return false;
15 }

17 int main() {
     ifstream fs("ospd.txt");
19   if (!fs) cout << "failed.";
     string werd;
21   cout << "Enter a word to see if it's an English word:\n";
     while (std::cin >> werd) {
23     fs.clear();
       fs.seekg(0);
25     cout << werd << " is";
       if (!isWord(werd, fs))
27         cout << " not";
       cout << " an English word.\n\n";
29     cout << "Enter a word to see if it's an English word:\n";
     }
31 }
```

(a) Describe what happens on line 18. What is the name of the variable desclared? What type of variable is it? How is it initialized?
ANS: The variable `fs` is an input file stream initialized to point to the first byte (character) of the file "ospd.txt," which, presumably, exists in the local directory.

(b) How could the boolean value `!fs` on line 19 turn out to be `true`?
ANS: The `ifstream` class inherits from `istream` and has following member functions to check for specific states of a stream (all of them return a bool value):

- `bad()` returns `true` if a reading or writing operation fails. For example, in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.
- `fail()` returns `true` in the same cases as `bad()`, but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.
- `eof()` returns `true` if a file open for reading has reached the end.

If any one of those returns `true` then `fs` becomes `false`.

(c) Why do we need the scope resolution for `namespace std` on line 22?
ANS: Name lookup is the procedure by which a name, when encountered in a program, is associated with the declaration that introduced it.

For example, to compile `std::cin >> werd;`, the compiler performs:

- unqualified name lookup for the name `std`, which finds the declaration of `namespace std` in the header `<iostream>`
- qualified name lookup for the name `cout`, which finds a variable declaration in the `namespace std`
- both argument-dependent lookup for the name `operator<<` which finds multiple function template declarations in the `namespace std` and qualified name lookup for the name `std::ostream::operator<<` which finds multiple member function declarations in `class std::ostream`

For function and function template names, name lookup can associate multiple declarations with the same name, and may obtain additional declarations from argument-dependent lookup. Template argument deduction may also apply, and the set of declarations is passed to overload resolution, which selects the declaration that will be used. Member access rules, if applicable, are considered only after name lookup and overload resolution.

For all other names (variables, namespaces, classes, etc), name lookup must produce a single declaration in order for the program to compile. Lookup for a name in a scope finds all declarations of that name, with one exception, known as the "struct hack" or "type/non-type hiding": Within the same scope, some occurrences of a name may refer to a declaration of a class/struct/union/enum that is not a typedef, while all other occurrences of the same name either all refer to the same variable, non-static data member (since C++14), or enumerator, or they all refer to possibly overloaded function or function template names. In this case, there is no error, but the type name is hidden from lookup (the code must use elaborated type specifier to access it). Go figure!

(d) What is happening on line 23 and why would we want to do that?
ANS: Eventually the `std::cin >> werd` command will lead to an `eof()` state that will put `cin` into a `fail` state. We want to `clear()` that flag this way.

(e) What is happening on line 24 and why would we want to do that?
ANS: We haven't closed and reopened the file, "ospd.txt" at this stage (which would accomplish the same thing as lines 23 and 24), we've just cleared the `fail` flag. To reposition the input file reader at the begining we invoke the `fs.seekg(0)` command.

(f) Assume that `ospd.txt` is a space-separated list of all the words (no proper nouns or punctuated words) of the English language. Further assume that `isWord()` defined here works as its pre-and postcondition specifications say. Does it require input to be in a special form to work correctly? That is to check if what the user types in is an English word or not?
ANS: Yes, the word must use only lower-case letters.

4. Write a function `bool isShrinkingWord(string word, ifstream& words);` that uses the `isWord()` function defined in problem 3 and returns `true` if either the word with its first character removed is also an English word or if the word with its last character removed is also an English word. For example `Pirate` is still a word (irate) if you remove its first letter and `rate` is still a word if you remove its last letter. So `isShrinkingWord("pirate", fs)` would return `true` (provided `fs` is referencing the first word in the list of all English words), and so would `isShrinkingWord("rate", fs)`. This function could be recursive, but it doesn't have to be.
ANS: This implementation requires many file resets...is there a better way? Can you recurse it?

```
bool isShrinkingWord(string& w, ifstream& fs) {
  cout << "Check if " << w.substr(1,w.size()-1) << " is a word.\n";
  cout << "isWord(" << w.substr(1) << ", fs)=";
  fs.clear();
  fs.seekg(0);
  cout << isWord(w.substr(1), fs) << '\n';
  fs.clear();
  fs.seekg(0);
  if (isWord(w.substr(1),fs)) {
    cout << w.substr(1) << " is a word.\n";
    fs.clear();
    fs.seekg(0);
    return true;
  }
  else if (isWord(w.substr(0, w.size() - 2),fs)) {
    fs.clear();
    fs.seekg(0);
    cout << w.substr(0, w.size() - 2) << " is a word." << '\n';
    return true;
  }
  else {
    fs.clear();
    fs.seekg(0);
    return false;
  }
}
```

5. Consider the following implementation of the Euclidean algorithm:

```
int gcd(const int& a, int b) {
  cout << a << ',' << b << '\n';
  if (b == 0) return a;
  return gcd(b, a%b);
}
```

(a) What is output by the function after the function call `gcd(2 * 2 * 3 * 5 * 11, 2 * 3 * 3 * 7 * 7);` ?
ANS: gcd(660,882)=gcd(882,660)=gcd(660,222)= gcd(222,216)=gcd(216,6)=gcd(6,0)=6, so the output is

```
660,882
882,660
660,222
222,216
216,6
6,0
6
```

(b) Explain what it means that the first parameter is a constant reference to an `int`.
ANS: A constant-reference is an address that the compiler won't allow to be changed in the body of the function, gcd().

(c) Explain why, if we changed the parameter list to `int gcd(int& a, int b)`, then a call to

```
gcd(2 * 2 * 3 * 5 * 11, 2 * 3 * 3 * 7 * 7);
```

produces the error: `'int gcd(int &,int)': cannot convert argument 1 from 'int' to 'int &'`
whereas if you first declare

```
int x = 2 * 2 * 3 * 5 * 11, y = 2 * 3 * 3 * 7 * 7;
```

then a call to `gcd(x,y)` produces the same output as what you got in part (a).
ANS: Because the integer literals can't be passed by non-const reference. It makes no sense if the callee could change a literal. Either pass by value (preferred here) or by const reference. When dealing with references you must consider two issues that are not present with values: lifetime and aliasing. A better word for const here might be "read-only."

6. Consider the complete program defined below:

```
1  using std::cin;
   using std::cout;
3  #include <vector>;
   using std::vector;
5
   int gcd(int& a, int b) {
7    return (b == 0) ? a : gcd(b, a%b);
   }
9
   class Rational {
11   int den; //denominator
     int num; //numerator
13 public:
     Rational(int n=0, int d=1) : num(n / gcd(n, d)), den(d / gcd(n, d)) {}
15   Rational operator*(const Rational& rho);
     Rational operator/(const Rational& rho);
17   Rational operator+(const Rational& rho);
     Rational operator-(const Rational& rho);
19   int getDen() const { return den; }
     int getNum() const { return num; }
21 };

23 fostream& operator<<(ostream& os, Rational& r) {
     os << r.getNum() << '/' << r.getDen();
25   return os;
   }
27
   Rational Rational::operator*(const Rational& rho) {
29   return Rational(num*rho.getNum(), den*rho.getDen());
   }
31
   int main() {
33   Rational r1(2 * 3, 3 * 5);
     cout << "r1 = " << r1 << '\n';
35   Rational r2(r1);
     r1 = r2 * r2;
37   Rational r3 = r1 * r2;
     cout << "r1 = " << r1 << '\n'
39     << "r2 = " << r2 << '\n'
       << "r3 = " << r3 << '\n';
41   Rational r4;
     cout << "r4 = " << r4 << '\n';
43   cin.get();
   }
```

(a) What is the default constructor here?
ANS: `Rational(int n=0, int d=1) : num(n / gcd(n, d)), den(d / gcd(n, d))` defaults to `Rational(0,1` if a variable is declared as `Rational` without being initialized.

(b) On what line(s) is a copy constructor called?
ANS: Lines 35, 36 and 37 all call one form or another of the copy constructor. Note that a `Rational` object is never passed by value here, which would entail another instance of a copy constructor. Basically, these instances are where an object of type `Rational` is constructed from another object of type `Rational`.

(c) What is the output of this program?
ANS: The program constructs and prints the `Rational r1 =` 2/5 (note the gcd eliminates the common factor, 3) and then clones `r1` as `r2` with a copy constructor (provided by default) and then writes over `r1` with the square of `r2` and prints out the new value of `r1`. Then `Rational r3` is created and assigned the value `r1 * r2` (note the use of the overloaded multiplication operator) which will be 8/125 = 2/5 * 4/2, at which point `r1 = 4/25`, `r2 = 2/5` and `r3 = 8/125` are all printed to the console with line feeds separating them. Finally, the default constructor is called produce `r4 = 0/1` and that's printed to the screen, yielding the output:

```
r1 = 2/5
```

```
r1 = 4/25
r2 = 2/5
r3 = 8/125
r4 = 0/1
```

(d) Write code to overload the + operator.

```
Rational operator+(Rational& rhs) {
        return Rational(numerator*rhs.getDen()+denominator*rhs.getNum(),
                        denominator*rhs.getDen());
}
```

(e) Write a function `vector<Rational> Fribonacci(unsigned n, Rational& r1, Rational& r2)` that will build a vector, `vr` of $n+2$ Rational numbers where `vr[0]=r1`, `vr[1]=r2` and `vr[i]=vr[i-1]+vr[i-2]` for `i>1`.
ANS: Here's one way to do:

```
#include "Rational.h"

vector<Rational> frational(unsigned sz, Rational x, Rational y) {
  vector<Rational> vr;
  vr.push_back(x);
  vr.push_back(y);
  for (unsigned i = 2; i < sz; ++i)
    vr.push_back(vr[i - 1] + vr[i - 2]);
  return vr;
}

int main() {

  Rational f0, f1;
  unsigned sz;
  cout << "Enter the first and second rational numbers in the form a/b: ";
  cin >> f0 >> f1;
  cout << "\nYou entered " << f0 << ", " << f1 << '\n';
  cout << "How many fribs do you want?\n";
  cin >> sz;
  vector<Rational> vr = frational(sz, f0, f1);
  for (Rational r : vr) cout << r << " ";
  cin.ignore();
  cin.get();
}
```

And some sample output:

```
Enter the first and second rational numbers in the form a/b: 2/3 3/5

You entered 2/3, 3/5
How many fribs do you want?
12
2/3 3/5 19/15 28/15 47/15 5/1 122/15 197/15 319/15 172/5 167/3 1351/15
```