Midterm Answers – CS 246 Winter 2009

Instructor: Ric Holt

February 26, 2009

These are not the only answers that are acceptable, but these answers come from the notes or class discussion.

1. (a) **1 mark** False

   (b) **1 mark** False

   (c) **3 marks**

   ```
   int i;
   i = 0;          // Can combine with previous line
   while ( i < LIMIT ) {
       cout << i << endl;
       i += 1;  // Or any other way to increment i by 1
   }
   ```

   0: no answer; 1: rough sketch; 2: almost right; 3: correct

   (d) **3 marks**

   ```
   switch ( i ) {
     case 0:
       cout << "zero";
       break;
     case 5:
       cout << "five";
       break;
     default:
       cout << "No go";
   }
   ```

   0: no answer; 1: rough sketch; 2: almost right; 3: correct

2. (a) **1 mark** True

   (b) **3 marks**

   ```
   cout << rand() % 3 + 2;    // Do not insist upon using srand
   ```

   0: no answer; 1: rough sketch; 2: almost right; 3: correct

   (c) **3 marks** It is a statement of what is assumed to be true when the associated function is called. It used used to help give the specification (or meaning) of the function. POSSIBLY: It helps in showing that the function is correct.

   3: both statements; 2: one statement; 1 rough idea; 0: no answer

   (d) **3 marks** This is a particular way to pass an (actual) argument to a (formal) parameter. The actual value of the argument is passed into a slot (or space or variable) within (the stack frame of) the called function.

   3: both stmts; 2: one statement; 1 rough idea; 0: no answer

3. (a) **3 marks** An assert or assertion statement takes an argument with evaluates to true or false. It is used to document and check the correctness of programs. It records assumptions about the state (of variables and constants) (in a particular location) in the program.

   1 mark for each of the above; total 1 mark for rough idea

   (b) **3 marks**

```
double ave( double n1, double n2, double n3 )
  {   return (n1 + n2 + n3) / 3.0;
```

   3: correct; 2: close; 1 rough idea; 0 no answer

   (c) **1 mark** True

```
double d[4];
```

   (d) **3 marks**

```
double total( double a[], int size ) {
     double sum = 0.0;   // Be sure that sum is explicitly initialized
     for ( int i = 0; i < size; i += 1 ) {
          sum += a[i];
     }
     return sum;
}
```

   3: correct; 2: just off no much; 1: rough idea; 0 : no answer

4. (a) **1 mark** False

   (b) **3 marks** Abstract Data Type. It is abstract: Defines a high level type, higher than the level of the source language. It defines data (as opposed to algorithm or function). It is a type, meaning that instances can be made of it (it can be instantiated).

   3: all 4 above; 2: 3 of above; 1: rough idea; 0 no answer

   (c) **3 marks** Arrays are more efficient than vectors Vectors are safer (and as well, easier to use) (could be explained in terms of crashes). Vectors are flexible (size can change dynamically). POSSIBLY: Notation for arrays is clearer and simpler Both implement (approximately) the same abstraction (sequence of indexed values)

   3 Both of top 3 statements; 2: 2 of top two; 1: rough idea; 0: no answer

   (d) **5 marks**

```
#include <iostream>
using namespace std;

#define SIZE 11            // The size should be some kind of constant
main() {
     // Can use either array or vector of strings
     string name[SIZE];    // Use string not C-string (char)
     for ( int i = 0; i < SIZE; i += 1 )
          cin >> name[i];
     for ( int i = SIZE - 1; i >= 0; i -= 1 )
          cout << name[i] << endl;
}
```

   5 Correct; 4: General algorithm good; 3: Seemed to understand; 1-2: rough idea; 0: no answer

5. (a) **3 marks** It is an object, part of whose value lies outside of the object, for example, in the heap. Explicitly implement:

- operator '=' overloading [Given]
- destructor
- copy constructor

3: All above; 2: 2 of above (not #1); 1: 1 of above (not #1); 0: no answer

(b) **3 marks** It is a constructor that has exactly one parameter, that is of the same type (class) as the (surrounding) class. That parameter must be passed by reference. Gets called:

- for value parameters
- for return values from functions
- for initialization

3: Any 3 of the 4 items above; 2: any 2 above; 1 rough idea; 0 no answer

(c) **3 marks**

```
double **M;  // Student can use any type, e.g., 'double' for basis of M
typedef double *ptrToDouble;
M = new ptrToDouble[N];
for ( int i = 0; i < N; i += 1 )
    M[i] = new double[N];
// ... at this point the matrix M[i][j] is ready to be used ...
```

3: Correct; 2: missed only simple points; 1: rough idea; 0: no answer

6. (a) **3 marks**

- Types : class, struct, enum, type alias
- function prototypes
- constants

3: 3 of these; 2: 2 of these; 1 rough idea; 0: no answer

(b) **3 marks #ifndef** is used to avoid having multiple copies of a header file in a compilation unit, due to **#include** from various files. Used as follows, for header file xxx.h

```
#ifndef XXX_H
#define XXX.H
// ... actual contents of the header file
#endif
```

Result is that a compilation unit actually gets a single copy or the contents of the header file.

3: Correct; 2: missing one things; 1: rough idea; 0: no answer

(c) **3 marks** An object module is the result of compiling (translating) a compilation unit (source code) to machine code, but not let linking the code into executable code. (For source file, such as xxx.cpp, the corresponding object module is usually called xxx.o.)

3: Correct; 2: Missing 2 thing; 1: rough idea; 0 : no answer

(d) **3 marks** Assume we have included a file such as iostream, **#include** <iostream>, we want to access item cout from the std namespace. Do it one of these 3 ways:

```
• using namespace std;              // Accesses all names in std namespace
• using std::cout;                  // Use just cout from std
• std::cout < "Output stuff";       // Each use of cout is prefixed by std::
```

7. (a) **2 marks** Names in the global name space are visible to all object modules being linked together. Names in a nameless namespace can only be accessed in its own compilation unit. (So, the global namespace is seen everywhere while each nameless name space is seen only in its own compilation unit.)

3: both of 2 statements; 2 1 of statements; 1: rough idea; 0: no answer

(b) **3 marks**

```
#include <iostream>
#include <fstream>
using namespace std;

int main() {
    ofstream outStream;
    outStream.open( "robert.txt" );
    string HelloString = "Hello";
    string BobString = "Bob";
    outStream << HelloString << endl;
    outStream << BobString << endl;
}
```

3: a correct program; 2: mostly correct; 1: rough idea; 0: no answer

8. **6 marks**

```
polynomial polynomial::operator *( const polynomial &rtSide ) {
    polynomial result;             // Internal vector automatically set to null
    if ( v.size() == 0 || rtSize.v.size() == 0 )
        return result;             // Which is null
    // assert neither side is null
    result.v.resize( v.size() + rtSize.v.size() ); // Elements set to zero
    for ( int i = 0; i < v.size(); i += 1 )
        for ( int j = 0; j < rtSide.v.size(); j += 1 )
            result.v[i + j] += v[i] * rtSize.v[j];
    return result;
}
```