University of Waterloo

Midterm Examination
Fall TERM 2009

Computer Science 246
Software Abstraction and Specification
Sections 01

Start Time: 16:30 End Time: 18:20
Number of Exam Pages (including cover sheet): 5
Total number of questions: 8
Total marks available: 86

CLOSED BOOK, NO ADDITIONAL MATERIAL ALLOWED

Instructor: Peter Buhr

October 27, 2009

Be brief, but you must answer the question! Do not rephrase the question in your answer; use point-form when possible. PROGRAMS DO NOT REQUIRE COMMENTS.

1. (a) **2 marks** What is a *shell*?

   (b) **3 marks** How are *file names* organized? Give a very brief description.

   (c) **2 marks** What is a *hidden file* and why are they hidden?

   (d) **3 marks** Write a globbing pattern that matches file names that do NOT start with the characters x, y, or z, followed by any number of characters, and ending with the suffix ".cc".

   (e) **3 marks** With respect to file permissions, what groupings can users be organized into, and what access control is allowed for users in each group?

2. (a) **1 mark** What is unusual about the basic type **int** in C/C++?

   (b) **2 marks** Give the values in each variable after the following code fragment has executed:

```
int i = 3; . . .
{ int k = i, i = 4; . . .
    { int i = i; // GIVE VALUES OF VARIABLES HERE
```

   (c) **3 marks** What are the 3 fundamental drawbacks of C strings?

   (d) **2 marks** How does the C++ string type eliminate the drawbacks in question 2c.

   (e) **2 marks** What is a *designated constant*? Give 2 examples.

3. (a) **2 marks** Variables p1 and p2 are pointers. In general, why is this assignment ambiguous?

```
p1 = p2;
```

   (b) **6 marks** Draw a picture of the stack at the point where rtn is about to return.

```
int rtn( int *&r ) {
    int i = 7, *pr = r;
    r = &i;
    // STATE OF STACK HERE
}
int main() {
    int i = 3, *pi = &i;
    rtn( pi );
}
```

   (c) **2 marks** Explain what is wrong with this code fragment and how to fix it:

```
int *parr = new int[10];
. . .
delete parr;
```

   (d) **1 mark** Which is more efficient: allocating a variable on the stack or in the heap?

4. (a) **2 marks** Give the precise execution meaning of operators such as "+=".

   (b) **2 marks** What is a *coercion* and why is it bad?

   (c) **2 marks** What is *short-circuit* expression evaluation?

   (d) **1 mark** What can be eliminated from programs by using the *modified version* of structured programming?

   (e) **2 marks** Explain what the **#define** preprocessor statement does, and what happens after it is encountered.

5. (a) **2 marks** Explain the difference between *formatted* and *unformatted* I/O.
   (b) **1 mark** True or False : There is an end-of-file character.
   (c) **2 marks** Explain the difference between *pass by value* and *pass by reference*.
   (d) **2 marks** How do *routine pointers* generalize a routine?

6. **10 marks** Write a shell script that compares the output (both standard out and standard error) from two programs given a list of test input-files. Print "identical output" (all lower-case letters), if the output from the two programs in the same for a specific input-file; otherwise print all differences in the output for each input file. The script has the following interface:

```
regress program1 'program1-options' program2 'program2-options' input-file-list
```

For example,

```
% regress cat '' cat '' input
identical output
```

compares the output from commands cat on input file input and prints "identical output".

```
% regress cat '' cat '-n' input
1,4c1,4
< abc abc
< abc ab
< ac
< a
---
>      1     abc abc
>      2     abc ab
>      3     ac
>      4     a
```

compares the output from commands cat, one of which adds line numbers to its output, on input file input and prints the differences.

```
regress a.out '4 5' a.out '4 5' input
```

compares the output from commands a.out on input file input and prints any differences.

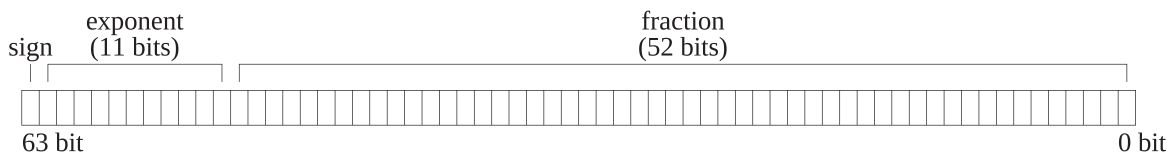
```
% regress regress "cat '' cat '-n'" regress "cat '' cat '-n'" input1 input2
identical output
identical output
```

recursively compares the output from the commands regress on the output from commands cat on input files input1 and input2 and prints "identical output" twice.

Use the UNIX command diff to compare and print differences in files.

NO ERROR CHECKING OF ANY FORM IS REQUIRED!

7. **16 marks** Write a **COMPLETE** C++ program to read in a file containing IEEE **double** values in internal format, and print out the value, and its sign bit, exponent bits, and fraction bits in hexadecimal. The IEEE floating point format is:



Generate output like the following, with each "title:value" pair separated by a tab character.

```
d:-3.5e-23       sign:1   exp:3b4  frac:527fcd8105c07
d:6.216e-15      sign:0   exp:3cf  frac:bfe8f60eb97b2
d:1.10396e-06    sign:0   exp:3eb  frac:2857a9da642ea
d:196.064        sign:0   exp:406  frac:88208d9427bfb
d:3.48209e+10    sign:0   exp:422  frac:36f94978d4fe
d:6.18419e+18    sign:0   exp:43d  frac:574aa19410800
```

Your program MUST use bit fields and **union** aggregation.

The shell interface to the program is as follows:

```
floatpoint input-file
```

The input file contains the **double** values in internal format.

NO ERROR CHECKING OF ANY FORM IS REQUIRED!

8. **10 marks** Write a C++ routine to parse a preprocessor **#define** directive to extract its name and value. The routine interface is:

```
bool define( string line, string &name, string &value );
```

Parameter line is an *input* parameter containing an entire line read from an input file for a C/C++ program. Parameter name is an *output* parameter containing the name of the define directive. Parameter value is an *output* parameter containing the value to replace the name. If a **#define** directive is present in the line, assume the **#define** directive is correctly formed, so the following simple regular expression can be used to parse it:

- the string `"#define"` starts at the beginning of the line,
- followed by one or more blank/tab characters,
- followed by a name composed of one or more characters NOT a blank/tab,
- followed by one blank/tab character,
- followed by a value composed of zero or more characters of any kind.

The routine returns **true** if a **#define** directive is present in the line, and returns its name and value through the output parameters. The routine returns **false** if a **#define** directive is NOT present in the line, and returns the empty string, `""`, for the name and value.

NO ERROR CHECKING OF ANY FORM IS REQUIRED!

The C++ string type stores arbitrary length sequences of characters with high-level operations to manipulate strings of characters.

```
string a, b, c;          // declare string variables
cin >> c;                // read white-space delimited sequence of characters
getline( cin, c, '\n' ); // read remaining characters until newline (newline is default)
cout << c << endl;       // print string
a = "abc";               // set value, a is "abc"
b = a;                   // copy value, b is "abc"
c = a + b;               // concatenate strings, c is "abcabc"
if ( a == b )            // compare strings, lexigraphical ordering
string::size_type l = c.length(); // string length, l is 6
char ch = c[4];          // subscript, ch is 'b', zero origin
c[4] = 'x';              // subscript, c is "abcaxc", must be character constant
string d = c.substr( 2, 3 ); // extract starting at position 2 (zero origin) for length 3, d is "cax"
c.replace( 2, 1, d );  // replace starting at position 2 for length 1 and insert d, c is "abcaxaxc"
string::size_type p = c.find( "ax" ); // search for 1st occurrence of string "ax", p is 3
p = c.rfind( "ax" );  // search for last occurrence of string "ax", p is 5
p = c.find_first_of( "aeiou" ); // search for first vowel, p is 0
p = c.find_first_not_of( "aeiou" ); // search for first consonant (not vowel), p is 1
p = c.find_last_of( "aeiou" ); // search for last vowel, p is 5
p = c.find_last_not_of( "aeiou" ); // search for last consonant (not vowel), p is 7
```

The C++ string *find* members return string::npos if a search is unsuccessful.