Midterm Examination of Introduction to Programming (CS1355-01)

November 28, 2011

1. (3%) Which one of the following statements is not equal to the other two (assuming that the loop bodies are the same)?

   (a) `do {...} while (i < 8);` (b) `for (; i < 8;) {...}` (c) `while (i < 8) {...}`

2. (3%) What is the result of the following program?

```
i = 5;
while (i > 0);
{
  printf("%d\n", i);
  i--;
}
```

3. (3%) What is the difference between the results obtained by the following two programs?

```
(a)                              (b)
   n = 0;                           sum = 0;
   sum = 0;                         for (n = 0; n < 8; n++) {
   while (n < 8) {                    scanf("%d", &i);
     scanf("%d", &i);                 if (i == 0)
     if (i == 0)                        continue;
       continue;                      sum += i;
     sum += i;                      }
     n++;
   }
```

4. (3%) Which of the following declarations are used to declare an unsigned short integer?

   (a) `unsigned short int` (b) `short unsigned int` (c) `int unsigned short`

   (d) `unsigned short`

5. (3%) Suppose that `c` is a variable of type `char`, `i` is a variable of type `int`, `f` is a variable of type `float` and `d` is a variable of type `double`. Explain what type conversions take place during the execution of each of the following statements.

   (a) `i = i + c;`

   (b) `f = f + i;`

   (c) `d = f + i;`

6. (3%) Explain why it is a good idea to append the `f` suffix to a floating-point constant if it will be assigned to a `float` variable (e.g., `f = 3.14159f`, where `f` is a `float` variable).

7. (3%) Explain why overflow will occur when the following program is executed in a 16-bit machine. And also describe how to avoid this overflow problem.

```
long i;
int j = 1000;
i = j * j;
```

8. (3%) Does the following statement always compute the fractional part of `f` correctly (assuming that `f` and `frac_part` are `float` variables)? If not, what is the problem?

```
frac_part = f - (int) f;
```

9. (3%) Use `typedef` to create a type named `Int8` and also define this type so that it represents 8-bit signed integers on a 32-bit machine.

10. (3%) Find the error in the following program and fix it.

```
int a[10], i;
for (i = 1; i <= 10; i++)
  a[i] = 0;
```

11. (3%) Find the problem in the following declaration and fix it.

```
int a[8] = {0, 0, 0, 0, 0, 0, 0, 0, 0};
```

12. (3%) The Fibonacci numbers are 0, 1, 1, 2, 3, 5, 8, 13, ..., where each number is the sum of the two preceding numbers. Write a program fragment that declares an array named `fib_numbers` of length 20 and fill the array with the first 20 Fibonacci numbers.

13. (3%) Find the error in the function definition below and fix it.

```
double average(double a, b) {
 return (a + b) / 2;
}
```

14. (3%) Find the error in the following program and fix it.

```
#include <stdio.h>
int main(void) {
  double x = 3.0;
  printf("Square: %d\n", square(x));
  return 0;
}
int square(int n) {
  return n * n;
}
```

15. (3%) Find the problem in the function definition below and fix it.

```
int power(int x, int n) {
  return power(x, n - 1);
}
```

16. (3%) Which of the following statements would be valid prototypes for a function that returns nothing and has one `float` parameter?

    (a) `f(float x);` (b) `void f(x);` (c) `void f(float);` (d) `void f(float x);`

17. (52%) Determine whether the following statements are correct or not (i.e., true or false). If not, please explain your reason (no reason, no point).

    (a) (2%) Once the compiler has seen `/*` symbol, it reads and ignores whatever follows until it encounters the next `*/` symbol.

    (b) (2%) Attempting to access the value of an uninitialized variable may yield an unpredictable result.

    (c) (2%) The following identifiers are all legal: (1) `Firth-exam` (2) `firth_exam` (3) `1th_exam` (4) `_1th_exam`

    (d) (2%) `%6.5d` dispalys the number 1234 as “␣␣1234”, where ␣ denotes the space character.

    (e) (2%) `%10.2f` dispalys the number 1.234 as “␣␣␣␣␣␣1.23”.

    (f) (2%) When serving as the format strings of `scanf`, `"%d-%d"` and `"%d␣-%d"` are equivalent, while `"%f,%d"` and `"%f,␣%d"` are not.

    (g) (2%) “`i = j = k = 1`” is the same as “`((i = j) = k) = 1`”.

    (h) (2%) The statement “`-i = i + 1;`” is legal.

    (i) (2%) Assume that `i` and `j` are `int` variables. Then the output produced by the following statements is “`5 11 6`”. (4%)

    ```
    i = 10;
    j = 5;
    printf("%d ", ++i - j++);
    printf("%d %d", i, j);
    ```

    (j) (2%) The logical operators treat any nonzero operand as a true value and any zero operand as a false value.

    (k) (2%) Assume that `i` and `j` are `int` variables. The output produced by the following statements is `0`.

    ```
    i = 10; j = 5;
    printf("%d", !i < j);
    ```

    (l) (2%) The body of a `do` loop is always executed at least once.

    (m) (2%) The `break` statement in the following program will transfer control out of the `while` statement.

```
while (...) {
  switch (...) {
  ...
  break;
  ...
  }
}
```

(n) (2%) By default, floating constants are stored as single-precision numbers.

(o) (2%) Suppose that `i` is a variable of type `int`, `j` is a variable of type `long` and `k` is a variable of type `unsigned int`. Then the type of the expression `i + (int) j * k` is `int`.

(p) (2%) An array is a data structure containing a number of data values, all of which may have different types.

(q) (2%) After executing the following declaration, the elements of the array `c`, from `c[0]` to `c[9]`, have values `5`, `1`, `9`, `3`, `7`, `2`, `6` and `0`.

```
int c[10] = {5, 1, 9, [4] = 3, 7, 2, [8] = 6};
```

(r) (2%) Although we visualize two-dimensional arrays as tables, C stores them in row-major order, with row 0 first, then row 1, and so forth.

(s) (2%) The length of a variable-length array is determined when the program is compiled, not when the program is executed.

(t) (2%) Both the expressions `sizeof(a) / sizeof(a[0])` and `sizeof(a) / sizeof(t)` can be used to calculate the number of elements in the array `a`, where `t` is the type of `a`'s elements.

(u) (2%) Executing the `return` statement in a function causes the function to return to the place from which it was called.

(v) (2%) C permits the definition of one function to appear in the body of another.

(w) (2%) Specifying the return type of a function to be `void` indicates that the function will return a value of type `void`.

(x) (2%) If we don't want a function to return any value, then we can omit the return type when defining this function.

(y) (2%) In C, arguments are passed by value and, therefore, any changes made to the parameters during the execution of the function don't affect the corresponding arguments.

(z) (2%) The statement "`return expression;`" in `main` function is not equivalent to "`exit(expression);`".