Banner number: Name:

Midterm Exam

CSCI 2132: Software Development

March 4, 2019

Marks	
Question 1 (10)	
Question 2 (10)	
Question 3 (10)	
Question 4 (10)	
Question 5 (5)	
Question 6 (5)	
Total (50)	

Instructions:

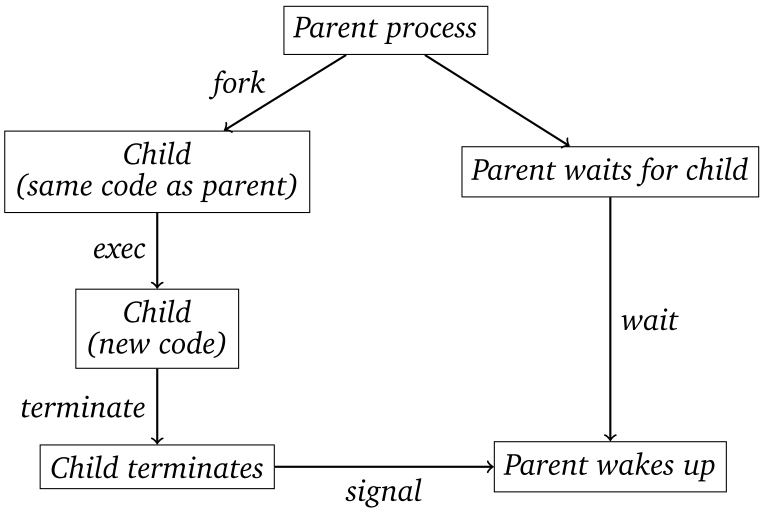
- **Write your banner number and name at the top of this page *before continuing to read.***
- **This exam has 10 pages, including this title page. Notify me immediately if your copy has fewer than 10 pages.**
- You are allowed to use **one cheat sheet, letter paper, both sides, 10pt font or larger.**
- Understanding the exam questions is part of the exam. Therefore, **questions will *not* be interpreted.** Proctors will only correct errors in question statements and clarify ambiguities, if any.

Question 1 (Processes) **10 marks**

(a) Put a check mark beside each true statement in the following list (and none beside each false statement):

On Unix, every process (except one) has exactly one parent process.	✓
The parent process of every process is a shell process.	
A process cannot have multiple threads.	
Creating and switching between processes is more costly than creating and switching between threads.	✓

(b) Draw a diagram that shows how a parent process starts a child process running a different program and waits for it to finish (as the shell does when you start a foreground process).



(c) Put a check mark beside each true statement in the following list (and none beside each false statement):

A foreground process can print to the terminal and read keyboard input.	✓
A background process can only print to the terminal but not read keyboard input.	✓
A program `prog` can be started in the background by typing `prog &` into the shell.	✓
A program `prog` can be started in the background by typing `prog` into the shell, then pressing `C-z`, and typing `fg` into the shell.	
`jobs` can be used to list all processes running on a Unix system.	
A job is a process started from the shell.	✓

Question 2 (C syntax and semantics) 10 marks

(a) Put a check mark beside each true statement in the following list (and none beside each false statement):

Most standard library functions are available to your program only after including an appropriate header file. ✓

The statement `#include <stdio.h>` includes the entire *text* of the `stdio.h` file in your program. ✓

C's `#include` statement and Java's `import` statement behave identically. ☐

(b) Put a check mark beside each true statement in the following list (and none beside each false statement):

If the `main` function returns an integer value, this value becomes the exit code of the program. ✓

Returning an integer from the `main` function is the *only* way to specify the exit code of a program. ☐

(c) In the following piece of C code, is the value of `x` well defined?

```
int a = 3, b;
int x = 2 * (b = a) + (a = 1);
```

If so, provide `x`'s value. If not, provide all possible values `x` could have.

No, `x`'s value is not well defined because the evaluation order of subexpressions in C is implementation-dependent.

If the left subexpression `b = a` is evaluated before the right subexpression `a = 1`, then `x`'s value is 7.

If the right subexpression is evaluated first, then `x`'s value is 3.

(d) What does the following piece of code compute?

```
int num;
for (num = 100; 1; ++num) {
  for (int div = 2; div < num; ++div) {
    if (!(num % div)) {
      goto label;
    }
  }
  break;
label: ;
}
printf("num = %d\n", num);
```

This computes the smallest prime number no less than 100.

(e) For both of the following two statements, state what the output is when they are executed.

```
int a = 0;
int b = 3;
if (a != 0 && b % a == 0) {
  printf("%d divides %d\n", a, b);
} else {
  printf("%d does not divide %d\n", a, b);
}
```

This prints "`0 does not divide 3`".

```
int a = 0;
int b = 3;
int non_null = a != 0;
int divides = b % a == 0;
if (non_null && divides) {
  printf("%d divides %d\n", a, b);
} else {
  printf("%d does not divide %d\n", a, b);
}
```

This throws a floating point exception (division-by-0 error).

Question 3 (C types) **10 marks**

(a) Put a check mark beside each true statement in the following list (and none beside each false statement):

C has multiple integer types. [✓]

C has only one floating point type. [ ]

C's character type can represent arbitrary UTF-16 characters. [ ]

(b) Why are the sizes and binary representations of integers and floating point numbers in C implementation-defined?

In order to allow the compiler to translate operations on number types directly to instructions available on the CPU.

(c) What does the following expression print?

```
if (a = 0) {
  printf("a is zero\n");
} else {
  printf("a is non-zero\n");
}
```

Does the output depend on the value of `a`? Justify your answer.

This always prints "`a is non-zero`", no matter what `a` is. The reason is that, instead of comparing `a` with 0, the conditional statement assigns 0 to `a` and the result (0) is used as the truth value of the condition, that is, the `else` branch is executed.

(d) Correct the code in question (c) so it does what it was most likely intended to do.

```
if (a == 0) {
  printf("a is zero\n");
} else {
  printf("a is non-zero\n");
}
```

(e) If a is a number between 0 and 99, what does the following code print?

```
char a1 = '0' + (a / 10);
char a0 = '0' + (a % 10);
printf("%c%c", a1, a0);
```

This prints the value of a *as a two-digit number.*

(f) Represent the following values in 16-bit 2's complement representation:

126	`0000 0000 0111 1110`
−9	`1111 1111 1111 0111`

What are the largest positive integer and the smallest negative integer that can be represented in 16-bit 2's complement representation?

Minimum	Maximum
−32,768	32,767

Question 4 (`printf` and `scanf`) **10 marks**

(a) List the conversion specifiers that need to be given to `printf` to

Print a string:	`%s`
Print an `unsigned long int`	`%lu`
Print an `int` using 10 digits, with leading zeroes if necessary:	`%010d`
Print a `double` using a width of 8 and 3 digits after the period:	`%8.3lf`

(b) If you try to read two integers using `scanf("%d%d", &x, &y)`, how do you check whether `scanf` succeeded in reading two values?

```
if (scanf("%d%d", &x, &x) != 2) {
  // Print an error message
}
```

(c) How do you adjust the `scanf` call in question (b) to figure out how many characters were read by `scanf`?

`scanf("%d%d%n", &x, &y, %n)` *ensures that* `n` *stores the number of characters that were read.*

(d) Are the following `scanf` invocations equivalent? If you answer no, provide an input that results in different values stored in `x` or `y` and explain the reason why the two `scanf` invocations behave differently for this input.

Are `scanf("%d%f", &x, &y)` and `scanf("  %d   %f", &x, &y)` equivalent?

Yes, they are equivalent.

Are `scanf("%d/%d", &x, &y)` and `scanf("%d / %d", &x, &y)` equivalent?

No, they are not equivalent.

On the input "`1  /  2`", the first `scanf` statement fails to read the input after `1` because the space does not match the expected character '/'.

The second `scanf` statement assigns `1` to `x`. Then it matches as many spaces as it reads to the space in the format. Then it matches the `/` in the input to the `/` in the format. It matches the spaces after `/` in the input to the space after `/` in the format. Finally, it reads `2` and assigns it to `y`.

(e) Does the following statement terminate when entering the string "`123`"?

```
scanf("%d ", &x);
```

Explain why not if your answer is no.

No, this does not terminate on this input.

The problem is that, after reading `123` and assigning it to `x`, `scanf` cannot decide how many spaces to skip corresponding to the space in the format string until it reads any character that is not a space.

Question 5 (Reading input and writing output) **5 marks**

Write a C program that does the following:

- Generate a random number between 1 and 100. (The code to do this is provided below.)
- Ask the user for a guess of the number (`What is your guess?`).
- If the guess is correct, print “`Correct`” and exit.
- If the guess is incorrect, print “`Too low`” or “`Too high`” and return to asking the user for another guess.
- The guessing process should loop until the user answers correctly or has made 8 guesses.
- If the user does not find the correct number after 8 guesses, the program should print “`Out of guesses. The correct number was XXX.`” and exit.

Your code does not have to perform any error checks (e.g., for ivalid input values).

```
#include <random.h>
#include <stdio.h>

int main() {
  int num, guess;
  srand();
  num = 1 + (rand() % 100);

  for (int i = 0; i < 8; ++i) {
    printf("What is your guess? ");
    scanf("%d", &guess);
    if (guess < num) {
      printf("Too low\n");
    } else if (guess > num) {
      printf("Too high\n");
    } else {
      printf("Correct\n");
      return 0;
    }
  }
  printf("Out of guesses.  The correct number was %d\n", num);
  return 0;
}
```

Question 6 (Manipulating numbers) **5 marks**

A decimal number is a palindrome if its digits read backward are the same as its digits read forward. Thus, 1, 33, 575, and 94322349 are all palindromes but 12, 12322, and 943359 are not.

Write a C program that does the following:

- Ask the user to enter a number (`Enter a number:`).
- Output "`This is a palindrome.`" or "`This is not a palindrome.`" depending on whether the given number is a palindrome.

Your code does not have to perform any error checks.

```
#include <stdio.h>

int main() {
  int num, low_digit, high_digit;

  printf("Enter a number: ");
  scanf("%d", &num);
  if (num > 0) {
    low_digit = high_digit = 1;
    while (high_digit <= num) {
      high_digit *= 10;
    }
    high_digit /= 10;
    while (high_digit > low_digit) {
      if ((num / low_digit) % 10 != (num / high_digit) % 10) {
        printf("This is not a palindrome.\n");
        return 0;
      }
      high_digit /= 10;
      low_digit *= 10;
    }
  }
  printf("This is a palindrome.\n");
  return 0;
}
```