Instructions:

- Print your name in the space provided below.
- This examination is closed book and closed notes, aside from the permitted one-page formula sheet. This examination is closed book and closed notes, aside from the permitted one-page fact sheet. Your fact sheet may contain definitions and examples, but it may not contain questions and/or answers taken from old tests or homework. You may include examples from the course notes.
- No calculators or other electronic devices may be used. The use of any such device will be interpreted as an indication that you are finished with the test and your test form will be collected immediately.
- Answer each question in the space provided. If you need to continue an answer onto the back of a page, clearly indicate that and label the continuation with the question number.
- If you want partial credit, justify your answers, even when justification is not explicitly required.
- There are 7 questions, some with multiple parts, priced as marked. The maximum score is 100.
- When you have completed the test, sign the pledge at the bottom of this page and turn in the test.
- If you brought a fact sheet to the test, write your name on it and turn it in with the test.
- Note that either failing to return this test, or discussing its content with a student who has not taken it is a violation of the Honor Code.

Do not start the test until instructed to do so!

Answers to questions are in blue.
Commentary (not expected as part of the answer) are in green.

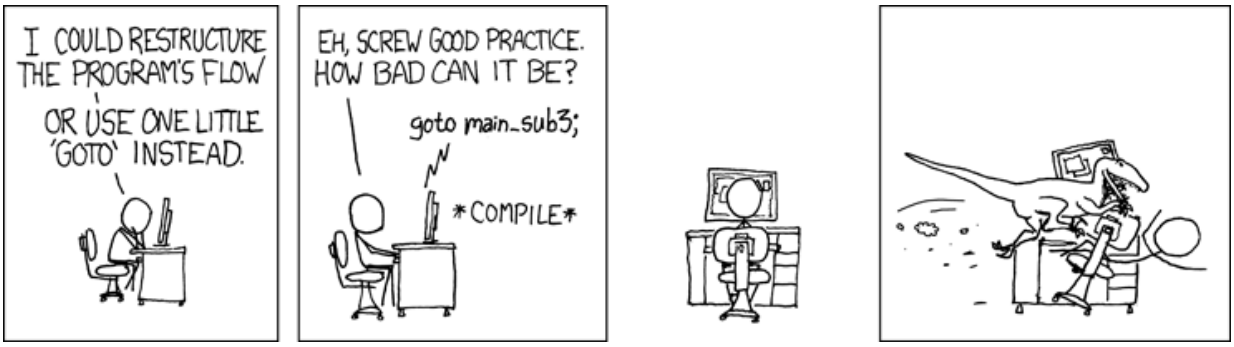
Name Solution

printed

Pledge: On my honor, I have neither given nor received unauthorized aid on this examination.

_____

signed


I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

xkcd.com

1. A CentOS user, `curunir`, executes the following commands in a bash shell:

```
1047 curunir in ~/orthanc/comm > ls -l
total 32
-rwxrw----. 1 curunir istari  13749 Sep 22 12:50 palantir
-rw-rw----. 1 curunir istari   6407 Sep 22 12:50 palantir.c
-rw-------. 1 curunir istari  23932 Sep 22 12:50 RingMemo.txt
-rw-r-----. 1 curunir istari  73429 Sep 22 12:50 MetalDetectors.odf
-rw-------. 1 curunir istari 984730 Sep 22 12:50 OrcBlueprints.odf

1048 curunir in ~/orthanc/comm > palantir
bash: driver: command not found...

1049 curunir in ~/orthanc/comm > PATH=./

1050 curunir in ~/orthanc/comm > palantir

1051 curunir in ~/orthanc/comm > ls -l
bash: ls: command not found...

1052 curunir in ~/orthanc/comm >
```

a) **[5 points]** Why did the command numbered 1048 shown above not cause `palantir` to be executed?

Since the error message says the command was not found, and we are in the directory containing palantir, and palantir is set to be executable for owner, it must be that the current directory is not in the user's path.

Permissions for the file palantir are correct. Permissions for other files are irrelevant.

b) **[5 points]** The command numbered 1050 did result in the execution of `palantir`. However, the command numbered 1051 resulted in an error, as shown. Why?

Command 1049 put the current directory in the path, so palantir was executed. But, command 1049 actually made the path contain ONLY the current directory.

When trying to find an executable, the shell will look in the directories in the user's path, in the order those directories are listed. The shell will not look anywhere else. So, if the shell reports a command cannot be found, it must be the case that there is no matching executable in any of the path directories.

c) **[5 points]** What command should the user have entered, instead of command 1049, in order to make command 1050 execute as shown, but also allow command 1051 to be executed correctly?

The user should have added the current directory, but not removed anything from the path: PATH=$PATH:./ will do it.

However, doing PATH=./:$PATH will make the current directory be the first one searched.

2. **[12 points]** Write a single command that the user `curunir` could execute in the directory shown above to create a tar file named `orb.tar`, located in his home directory, and containing only the `odf` files in his current working directory.

tar cf ~/orb.tar *.odf

You needed to:

- **specify the right switches; cf will do, cvf will also do**
- **specify the directory where orb.tar is to be created; ~ is a shortcut for your HOME directory**
- **specify that you wanted the tar file to include the odf files in the current directory; you could list them individually, or use a wild-card**

3. A novice C programmer writes the following short program, which is supposed to perform as described in the comments. The code compiles without any warning or error messages, even with the `-Wall` and `-W` switches.

```
#include <stdio.h>
void Swap(int A, int B);
int  W = 11;

int main() {

   int x = 53, y = 19;
   int sum = Swap(x, y);
   printf("x  :%3d\ny  :%3d\nsum:%3d\n", x, y, W);

   x += W;
   y += W;
   int sum = Swap(x, y);
   printf("x  :%3d\ny  :%3d\nsum:%3d\n", x, y, W);

   return 0;
}

/**  Swap the values of user's variables
 *   Pre:  A and B are initialized
 *   Post: the values in the user's variables have been swapped;
 *         W has been modified
 */
void Swap(int A, int B) {

   int tmp = B;
   A = B;
   B = tmp;
   W = A + B + W;
}
```

Relevant fact: has file scope, so accessible everywhere
Irrelevant fact: static storage duration

Relevant facts: no linkage, so inaccessible outside this function, unless passed by pointer
Irrelevant facts: automatic storage duration, local scope

Relevant facts: passed by value

Irrelevant facts: swap logic is incorrect, so A and B are not swapped

There is a logic bug in the body of Swap(); it actually makes A and B both take on the value of B. That can be fixed by assigning A to tmp instead of B.

There's also a bug in that Swap() is void, so it doesn't return a value, but the call in main() is in an assignment statement, so main() expects a return value. That was the result of an editing error; however, it had nothing to do with any of the questions below.

Since W is declared at file scope, Swap() has full access to W and can modify it. The code that updates W works just fine. W also has static storage duration (since it's declared at file scope), but that is entirely irrelevant to this question.

The essential fact about the situation in this question is that actual parameters (those used in the call to a function) are passed into the function by value. That is, the values of the actual parameters are copied into the formal parameters in the function definition. So, a function is always working with copies of the caller's variables, not with the caller's variables.

That means that Swap() will NOT swap the values of x and y. That would still be true if x and y were declared at file scope, as long as they were passed as parameters to Swap(), so the fact that x and y were local to main() is irrelevant (with the given implementation of Swap()). And, it makes no difference whether x and y have static storage duration. The issue is access.

a) **[4 points]** Exactly what will be printed after the first call to `Swap()`? Be very precise.

```
x   : 53
y   : 19
sum: 49
```

This just involves tracing the execution of the code. Aside from arithmetic errors, the common errors were to not understand that Swap() cannot actually modify x or y, and to overlook the logic error in Swap() that left A and B both equaling B (since that affected how W is updated).

We did not make any deductions based on whether your answer contained the right formatting.

b) **[4 points]** Exactly what will be printed after the second call to `Swap()`? Be very precise.

```
x   :135
y   :102
sum:320
```

This also just involves tracing the execution of the code. In addition to the errors noted above, some students did not take into account the two assignment statements that reset x and y before the second call to Swap().

c) **[8 points]** The given implementation of `Swap()` does not perform as the novice programmer intended (according to the header comment he wrote). Is there any way the programmer could make the implementation of `Swap()` perform correctly, by changing the C code in body of `Swap()` but not making any changes elsewhere? If so, show how it could be done. If not, explain why not.

The given implementation does not swap the values of the user's variables, x and y, because parameters are passed by copy, so Swap() cannot affect the values of x and y.

Changing only the body of Swap() cannot fix the problem. There's a version in the course notes that uses pass-by-pointer, and that does work, but that would require changing the calls to the function as well as the body.

Common errors included:

- citing the scope of x and y (local to main()) as the problem
- overlooking the fundamental issue that Swap() deals with copies of x and y
- being distracted by the logic bug inside Swap()
- suggesting a fix using pass-by-pointer, but not realizing that would require modifying both the declaration of Swap() and the call to Swap(), both of which are outside of Swap()

4. Consider the following C function:

```
int nextDivisor(int N) {

   static int divisor = 1;

   while ( divisor <= N ) {
      if ( N % divisor == 0 ) {
         int found = divisor;
         divisor++;
         return found;
      }
      divisor++;
   }
   return 0;
}
```

a) **[7 points]** When the function above is called as shown below, the values 1, 3, and 5 are printed. Why? Be sure to explain why the value 1 isn't printed every time.

```
printf("%5d", nextDivisor(15));
printf("%5d", nextDivisor(15));
printf("%5d", nextDivisor(15));
```

The variable divisor has static storage duration, so it retains its value from the previous call; therefore, each call starts by considering the next integer after the previous divisor that was found.

The key point is that divisor has static storage duration; without saying that, you cannot explain why the values 3 and 5 are printed if the given code is executed.

b) **[7 points]** Explain the logic of the C loop test shown below. Be precise. How many integer values will be printed?

```
int N = 12;
int divisor;

while ( ( divisor = nextDivisor(N) ) > 0 ) {
   printf("%d\n", divisor);
}
```

The loop test compares the result of an assignment operation to 0.

In C, the assignment statement has a value, namely the value that is copied to the variable on the left side of the assignment, which is the value of the most recent divisor.

The nextDivisor() function returns 0 when divisor becomes larger than N, so that terminates the loop; and nextDivisor() must eventually return 0 since divisor is incremented at least once on each call.

The six values printed are: 1, 2, 3, 4, 6, 12

The question was about the loop test. You needed to explain why the loop will terminate, and explain the semantics of using the assignment statement within the loop test. Almost no one did the latter, and failure to explain that cost you 1 point.

5. Suppose a user executes the following commands:

```
1138 CentOS > ls -l > out.txt
1139 CentOS > wc < out.txt
  18 155 1316
```

a) **[6 points]** Why is no output displayed after the execution of command 1138?

The > character is the output redirection operator, which redirects the normal `ls -l` output to the file `out.txt`. As such, the output no longer appears in the shell, and the user is prompted for their next command when the output is complete.

Common errors included interpreting > to mean the process is running in the background, as well as asserting that `ls -l` must be executed with a directory argument and cannot run on a file (correct, but missing the > operator).

b) **[6 points]** What is the meaning of the numbers following line 1139? If the `ls` command also included the `-a` switch, would these numbers be larger, smaller, or stay the same? Why?

`wc` executes the word count program. The < character is the input redirection operator, which passes the contents of out.txt into wc. The 18 represents the number of lines in `out.txt`, the 155 represents the number of "words" (for a loose definition of "word"), and the 1316 represents the number of characters (or bytes) in the file.

The `-a` switch displays all files in the provided directory, including any hidden files that exist, as well as listings for the current directory (.) and parent directory (..). Running both `-l` and `-a` will therefore generate more output than just `-l` alone, increasing the number of lines, words, and characters in out.txt.

Common errors included interpreting 18 as words, 155 as characters, and 1316 as bytes (or bits), missing the "also" in the question and answering for only `ls -a`, and incorrect answers for the behavior of the `-a` switch.

c) **[4 points]** Write a single command that produces the same final output as commands 1138 and 1139, but does not require `out.txt`.

```
ls -l | wc
```

To avoid using `out.txt`, we need the output of `ls -l` to flow directly into the `wc` program. This can be done by piping the output of `ls -l` into `wc` with the | operator, thereby chaining the two commands together.

Common errors included switching the order of the commands (`wc | ls -l`), attempting to use input/output redirection operators that require a file rather than a program (`wc < ls -l` or `ls -l > wc`), and running `wc` on `*.txt` (could produce different output if more than one `.txt` file is present in the working directory).

6. Consider the following Java method:

```
public static void main(String args[]) {
    double myVar = 1.618;
    char m;
    for (int i = 0; i < 2; i++)
        myVar /= 2;
    if (myVar < 0.5)
        m = 'a';
    else
        m = 'b';
    System.out.println("myVar: " + myVar + " m: " + m);
}
```

a) [6 points] Write a C `main()` function that is functionally equivalent to the Java method. Show any necessary include directives.

```
#include <stdio.h>

int main() {
   double myVar = 1.618;
   char m;
   for (int i = 0; i < 2; i++)
      myVar /= 2;
   if (myVar < 0.5)
      m = 'a';
   else
      m = 'b';
   printf("myVar: %f m: %c", myVar, m);
   return 0;
} //main
```

Due to the "functionally equivalent" phrasing in the question, there are a number of ways to answer this question. Acceptable modifications to the above code include (but are not limited to):

- Using a `while` loop instead of a `for` loop
- Initializing `char m = 'a';` and only updating it to `b` in a single `if`, or initializing `char m = 'b';` and only updating with to `a` in a single `if`
- Making `main()` a `void` function and not returning 0
- Skipping the division `for` loop entirely and initializing `double myVar = 1.618/4;`

Modifications that were not acceptable and common mistakes include:

- Using integers (or `int_32` datatypes) for `myVar` and `char` (and using `%d` format specifiers)
- Using `println()` and Java syntax instead of `printf()` and C syntax
- Forgetting `#include` directives
- Placing the `if` blocks inside of the `for` loop
- Missing closing curly braces

b) **[4 points]** Demonstrate how the code can be adapted to accept a starting value for `myVar` from the shell. In C, a string may be converted into a `double` with the `atof()` function, which takes the form `double atof (const char* str)`.

```
int main (int argc, char *argv[]) {
    myVar = atof(argv[1]);
    // No other modifications to the code are necessary
```

The intent of the question was for you to provide the alteration above; however, ambiguity in the phrase (from the shell) made the use of `scanf()` an acceptable solution. You could read in a cstring and then use `atof` to convert it to a `double`, or read in a `double` directly.

Common mistakes included:

- Passing in only a `String` (or `String[]`) into `main()` (we did not clearly state that Strings don't exist in C until after the exam, but sample code in the lectures when discussing command lines parameters used `char*`)
- Using `argv[0]` rather than `arvg[1]`
- Including the datatypes in the `atof` conversion (e.g., `double myVar = double atof(const char* argv[1]);`)
- Attempting to chain together `scanf()` and `atof()` in methods that cause syntax and runtime errors (e.g., `double myVar = atof(scanf(%s, &str));`)
- Incorrect syntax for `scanf()` (e.g., `scanf("%s", var);` and `scanf("&var");`)

c) **[4 points]** Alter the `printf` command so that the output is displayed in the following format (nothing in the code besides the `printf` command should change).

```
myVar: 0.40450000
m: aaa
```

```
printf("myVar: %10.8f\nm: %c%c%c", myVar, m, m, m);
```

The two things that I was looking for here were the specified number of columns after the decimal point for `myVar`, as well as the use of 3 `%c` format specifiers and m variables in the command. Not including the 10 for `myVar` was OK (as was using some different values), as format specifiers `%.8f`, `%1.8f`, and `%5.8f` all will produce the same output as `%10.8f`.

Common mistakes included:

- Neglecting the number of columns for the `%f` format specifier
- Using `%d` for the floating point `myVar`
  - (I tried to only take off points for this once in question 6 (either in part A or part C) if you made the same mistake in both)
- Using `%3c` rather than `%c%c%c`

7. A programmer is testing her implementation of a simple game. The computer generates a random number, and the player attempts to guess the number correctly. In addition to the main function, the programmer's code includes the two additional functions declared here:

```
/**  Prompts the player to enter a guess and obtains the guess
 *
 *   Post: returns the player's guess to the calling function
 */
int promptGuess();

/**  Determines if the player's guess matches the computer's number
 *
 *   Pre:  receives the player's guess and the computer's number
 *   Post: prints a correct/incorrect message and returns true/false
 */
bool handleGuess(int, int);
```

The programmer writes the code, it compiles successfully, and she attempts to run the game for the first time. It does not go well (user input is formatted in **bold**):

```
1234 CentOS > gcc -o guessGame -std=c99 guessGame.c
1235 CentOS > ./guessGame
Ready for your next guess: 50
Segmentation fault (core dumped)
```

Segmentation faults are always frustrating. So, the programmer uses `gdb` to analyze her implementation:

```
1237 CentOS > gdb guessGame
. . .
(gdb) break promptGuess
Breakpoint 1 at 0x400745: file guessGame.c, line 32.
(gdb) run
Starting program: /home/jwenskovitch/examq/guessGame

Breakpoint 1, promptGuess () at guessGame.c:32
32            printf("Ready for your next guess: ");
(gdb) n
33            scanf("%d", g);
(gdb) n
Ready for your next guess: 50

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7a750d2 in _IO_vfscanf_internal (s=<optimized out>, format=<optimized out>,
    argptr=argptr@entry=0x7fffffffdce8, errp=errp@entry=0x0) at vfscanf.c:1826
1826                    *ARG (unsigned int *) = (unsigned int) num.ul;
(gdb) backtrace
#0  0x00007ffff7a750d2 in _IO_vfscanf_internal (s=<optimized out>, format=<optimized
    out>, argptr=argptr@entry=0x7fffffffdce8, errp=errp@entry=0x0) at vfscanf.c:1826
#1  0x00007ffff7a83b09 in __isoc99_scanf (format=<optimized out>) at isoc99_scanf.c:37
#2  0x0000000000400768 in promptGuess () at guessGame.c:33
#3  0x000000000040070f in main () at guessGame.c:16
(gdb) list guessGame.c:32
29
30     int promptGuess() {
31            int g;
32            printf("Ready for your next guess: ");
33            scanf("%d", g);
34            return g;
35     } //promptGuess
```

a) **[6 points]** The `gdb` session above tells the programmer there is a specific error in the code they have written. What is that error, what specific `gdb` output, other than C code, indicates the location and behavior of the error, and how can the error be fixed?

The error in the user's code is that they forgot the ampersand character before `g` in line 33.

This can be seen via the `backtrace` output, which shows line 33 in `guessGame` to be the location in the user code from which the error was generated, and then examining the C code provided by `list`.

To fix the error, update line 33 to read `scanf("%d", &g);`

Common errors included:

- **Only noting the error at line 1826 of `vfscanf.c` (This is in the library code, while the question asks about an error in the user's code)**
- **Asserting that the fix is to declare `g` as an `unsigned int` (Probably reasonable since the user isn't guessing negative numbers, but will not fix the issue)**
- **Asserting that `scanf()` must read a string, and then have that string converted to an integer (The `%d` format specifier reads an integer)**
- **Not answering all three parts of the question clearly**

Note that this example (a missing & in `scanf()`) was discussed in the gdb lecture.

The programmer fixes the rather embarrassing error mentioned above, recompiles the code and executes it again. This time the game executes successfully, until the game is won:

```
Sorry, your guess was too low.  Try again.
Ready for your next guess: 87
Sorry, your guess was too low.  Try again.
Ready for your next guess: 93
Hooray!  You guessed correctly!
Ready for your next guess:
```

Well, that is still wrong. The game should exit after the player guesses correctly, but instead it prompts the player for yet another guess. So, she resorts to `gdb` again:

```
(gdb) run
. . .
Nope, your guess was too high. Try again.
Ready for your next guess: 57
Nope, your guess was too high. Try again.
Ready for your next guess: 53

Breakpoint 2, handleGuess (guess=53, myNum=53) at guessGame.c:44
44                  printf("Hooray!  You guessed correctly!\n");
(gdb) n
Hooray!  You guessed correctly!
45                  return true;
(gdb) n
52      } //handleGuess
(gdb) n
main () at guessGame.c:15
15              while (!success) {
(gdb) n
16                  int g = promptGuess();
(gdb) n
Ready for your next guess: 53
18                  handleGuess(g, myNum);
```

```
(gdb) n

Breakpoint 2, handleGuess (guess=53, myNum=53) at guessGame.c:44
44                  printf("Hooray!  You guessed correctly!\n");
(gdb)
```

Puzzled by what gdb is showing, the programmer decides to display some of the relevant code:

```
(gdb) list guessGame.c:47
42  bool handleGuess(int guess, int myNum) {
43      if (guess == myNum) {
44              printf("Hooray!  You guessed correctly!\n");
45              return true;
46      } else if (guess > myNum) {
47              printf("Nope, your guess was too high. Try again.\n");
48      } else {
49              printf("Sorry, your guess was too low.  Try again.\n");
50      } //if-else
51      return false;
52  } //handleGuess
(gdb) list guessGame.c:18
13      bool success = false;
14
15      while (!success) {
16              int g = promptGuess();
17              handleGuess(g, myNum);
18      } //while
19
20      printf("Thanks for playing!\n\n");
21
22     return 0;
```

Still uncertain, the programmer checks the value of success after returning from the handleGuess function.

```
Breakpoint 1, handleGuess (guess=67, myNum=67) at testq.c:44
44              printf("Hooray!  You guessed correctly!\n");
(gdb) n
Hooray!  You guessed correctly!
45              return true;
(gdb) n
52  } //handleGuess
(gdb) n
main () at testq.c:15
15      while (!success) {
(gdb) print success
$1 = false
```

b) [7 points] The gdb session above tells her there is a specific error in the code she has written. What is that error, what specific gdb output, other than C code, indicates the location and behavior of the error, and how can the error be fixed?

The error in the user's code is that they never capture the return value from `handleGuess()`. As a result, the `success` variable never changes from `false`, and the `while` loop continues running infinitely.

This can be seen from the breakpoint at the beginning of `handleGuess()` showing that `guess` and `myNum` have the same values, the `list` command showing the full `handleGuess()` function returning a `bool` along all paths, the `list` command showing the relevant part of `main()` not storing the return value from `handleGuess()`, `handleGuess()` returning `true` on line 45 in the step through, and the `print` command showing that `success` remains `false` after the `handleGuess()` function returns `true`.

To fix the error, update line 17 to read `success = handleGuess(g, myNum);`

Common errors included:

- **Asserting that the error could be fixed by setting `success = true` instead of (or before) returning `true` in `handleGuess()` (The `success` variable is scoped to `main()` and cannot be accessed by `handleGuess()` without giving it file scope)**
- **Asserting that `handleGuess()` always returns `false` because a `return false;` command is at the end of the function (We see that it returns `true` in the step through)**
- **Not answering all three parts of the question clearly**