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 from the course notes, but it may not contain examples, questions and/or answers taken from old tests or homework.
- 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 5 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 are in blue

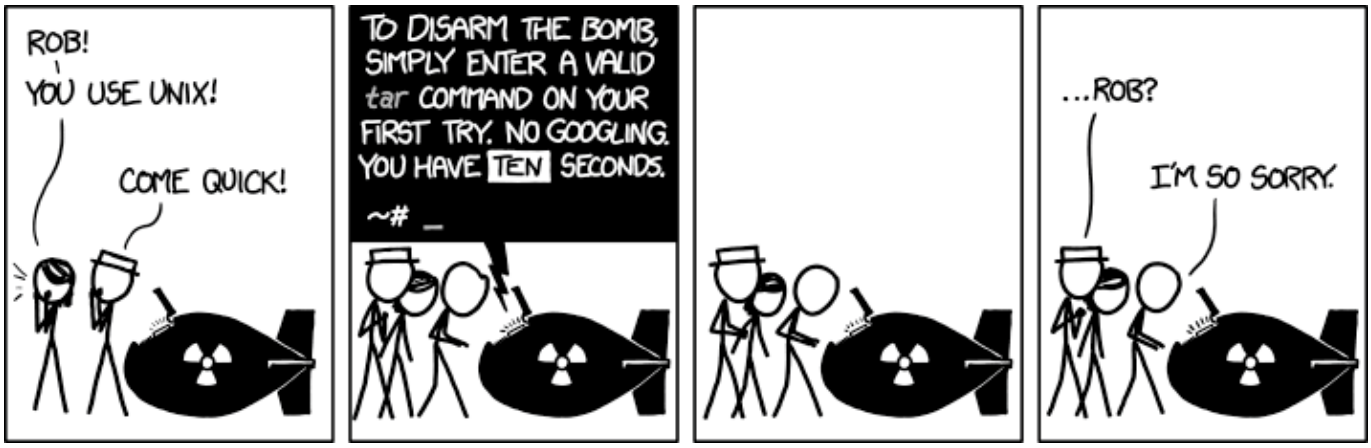
Commentary is in green

Name Solution

printed

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

signed


ROB!
YOU USE UNIX!
COME QUICK!
TO DISARM THE BOMB, SIMPLY ENTER A VALID tar COMMAND ON YOUR FIRST TRY. NO GOOGLING. YOU HAVE TEN SECONDS.
~# _
...ROB?
I'M SO SORRY.

xkcd.com

1. **[20 points]** Write an implementation of the following C function:

```
/*  Reverses the order of the elements in A.
 *
 *  Pre:   A[0:Sz-1] hold the values to be reversed
 *  Post:  The values in A[] have been reversed
 *
 *  Example:
 *     {17, 4, 22, 19, 41, 18, 30} -> {30, 18, 41, 19, 22, 4, 17}
 *
 *  Restrictions:
 *    - uses only variables that are parameters or declared within
 *      the function
 */
void reverse(int A[], int Sz) {

   for (int idx = 0; idx <= Sz/2; idx++) {

      int tmp = A[idx];

      A[idx] = A[Sz - 1 - idx];

      A[Sz - 1 - idx] = tmp;
   }
}
```

Here's a more elegant solution:

```
void reverse2(int A[], int Sz) {

   int lo = 0;            // index at low end of unswapped data
   int hi = Sz - 1;       // index at high end of unswapped data

   while ( lo < hi ) {    // keep going until lo and hi meet, or pass

      int tmp = A[lo];    // swap values at low and high ends

      A[lo] = A[hi];
      A[hi] = tmp;

      lo++;               // step toward middle,
      hi--;               //    from both ends
   }
}
```

One common error was to iterate too long, so that the values were reversed and then reversed again.

2. A programmer wants to implement the function `countMod4()`, as described in the header comment below. But his current implementation does not work as desired; it always prints the same value, 0.

```
uint32_t countMod4();

int main() {

   for (uint32_t i = 0; i < 9; i++) {

      fprintf(stdout, "%3"PRIu32":  %"PRIu32"\n", i, countMod4());
   }

   return 0;
}

/*   Computes values for a mod-4 counter, starting at zero.
 *
 *   Returns:
 *      When called successively, returns the values
 *         0, 1, 2, 3, 0, 1, 2, 3, . . .
 */
uint32_t countMod4() {

   uint32_t modder = 4;
   uint32_t counter = 0;

   return counter++ % modder;
}
```

The basic issue is that the variable `counter` is reset to 0 each time the function is called; we can fix that by giving `counter` static storage duration; there are two ways to accomplish that…

a) **[10 points]** Explain precisely how the programmer could correct the given implementation by moving ONE line of code that is currently within the body of the function `countMod4()` to a different location in the file, and making no other changes to the given code.

Move the declaration of the variable `counter` so that it is file-scoped.

Then `counter` will have static storage duration, so the changes made in one call will persist until the next call.

b) **[10 points]** Explain precisely how the programmer could correct the given implementation by changing ONE line of code within the body of the function `countMod4()`, and making no other changes to the given code.

Declare the variable `counter` to be static: `static uint32_t counter = 0;`

3. [20 points] Complete the implementation of the function described below:

```
/*   Returns the index of the last occurrence of a given character in a
 *   given C string.
 *
 *   Pre:   ch is initialized
 *          *str is a properly-terminated C string
 *   Returns:
 *      The index of the last occurrence of ch in *str;
 *      -1 if ch does not occur in *str.
 *
 *   Examples:
 *      'o' and "comp organization" -> 15
 *      'w' and "comp organization" -> -1
 */
int findLastOf(char ch, const char* str) {

   int matchIdx = -1;

   int idx = 0;

   while ( str[idx] != '\0' ) {

      if ( str[idx] == ch ) {

        matchIdx = idx;
      }

      idx++;
   }

   return matchIdx;
}
```

Some comments:

- you can't use strcmp() to compare a char to a string
- 'x' is a char; "x" is a string (char array); the difference matters
- '\0' is the string terminator; '/0' is not
- you can't assume ch will be found in the string

Here's a somewhat more efficient solution:

```
int findLastOf2(char ch, char* str) {

   int matchIdx = -1;
   int idx = strlen(str) -1;

   while ( idx >= 0 ) {

      if ( str[idx] == ch ) {
         return matchIdx;
      }
      idx++;
   }

   return matchIdx;
}
```

4. [25 points] In the following, you will see a series of pointer type questions. You may assume that the complete first statement in each part allocates an appropriate target or set of values for the variable `p`. Infer the most appropriate data type of the variable `ptr` based on the variable `p` and its type. Please write down the type of variable `ptr` right next to each code segment.

a)
```
int *p = …;
ptr = &p;
```
p is an int*, so &p is an int**.

Therefore, ptr should be an int**.

b)
```
int **p = …;
ptr = *p;
```
p is an int**, so *p is an int*.

Therefore, ptr should be an int*.

c)
```
char **p = …;
*p[1] = *ptr;
```
p is a char**, so p[1] is a char*, and *p[1] is a char.

Therefore, *ptr should be a char,

so ptr should be a char*.

d)
```
char p[3] = ..;
**ptr = &p[1];
```
p[1] is a char, so &p[1] is a char*.

Therefore, **ptr should be a char*,

so ptr should be a char***.

e)
```
char **p = ..;
*ptr = &p[1];
```
p is a char**, so p[1] is a char*, and &p[1] is a char**.

Therefore, *ptr should be a char**,

so ptr should be a char***.

5. [15 points] Consider the following short C program.

```
#include <stdio.h>

int y = 0;

void mystery(int num)
{
   static int x = 0;
   int z = 0;

   x += num;
   y += num;
   z -= num;

   printf("x is:  %d\n, y is:  %d\n, z is:  %d\n", x, y, z);
}


int main()
{
   mystery(15);   // call 1
   mystery(7);    // call 2
   mystery(25);   // call 3

        return 0;
}
```

Note that:
y has static storage duration, since it's file-scoped
x has static storage duration, since it's declared as static (local static)

What would be the output of running the C program?

From call 1:

Variable	before call	after call
x	0	15
y	0	15
z	0	-15

From call 2:

Variable	before call	after call
x	15	22
y	15	22
z	0	-15

From call 3:

Variable	before call	after call
x	22	47
y	22	47
z	0	-25