COS 217 Midterm Exam

STUDENT NAME

Search students by name or email...

Q1 Instructions and Pledge

2 Points

This exam consists of ***five*** multi-part questions (plus the pledge), and you have 60 minutes — budget your time wisely.

This is a partially "open-book" exam, with these ***allowances***:

- you may refer to the course's textbooks and the course materials found on or linked directly from the course's website.
- you may refer to your own notes and assignment solutions.
- you may use blank paper as scratch space, but you must enter your answer in the online system in order to receive credit.

In contrast to the allowances above, this exam has these ***restrictions***:

- You may **not** access other information on the Internet.
- You may **not** compile or run any code on `armlab` or any other machine.
- You are not allowed to communicate with any other person, whether inside or outside the class. You may not send the exam problems to anyone, nor receive them from anyone, nor communicate any information about the problems or their topics. Because students may be taking the exam late, this applies from when you take the exam until the sample solution has been published.
- The ***only*** form of allowable help from any person is that if you have technical issues or need to ask a clarifying question about the wording of some problem, you may post a **private** message on Ed for the course staff to answer at their discretion.

This examination is administered under the Princeton University Honor Code, and by signing the pledge below you promise that you have adhered to the instructions above.

Please type out the Honor Code pledge exactly as follows, including this exact spelling and punctuation:

> *I pledge my honor that I have not violated the Honor Code during this examination.*

Enter your answer here

Now type your name as a signature confirming that you have adhered to the Honor Code:

Enter your answer here

Save Answer

Q2 Teeny-tiny Computer

10 Points

Suppose we have a 6-bit computer. Answer the following questions.

Q2.1

2 Points

What is the **largest unsigned** number that can be represented in 6 bits?

In Binary:

Enter your answer here

In Hexadecimal:

Enter your answer here

Save Answer

Q2.2

2 Points

What is the **smallest** (i.e., most negative) **signed** number represented in 2's complement in 6 bits?

In Binary:

Enter your answer here

In Decimal:

Enter your answer here

Save Answer

Q2.3

6 Points

When doing **signed arithmetic** using 2's complement representation in 6 bits, consider the following three operations, where the first number is -12 and the second number is some *positive* number.

For each operation, indicate (Yes/No) whether it is possible for the result to **overflow**?

If an overflow is possible, provide an *example* that shows the second number in *decimal* (you don't need to show the result).
If an overflow is not possible, say "None" (no explanation is needed).

1. **Addition**

- Yes
- No

Example of second number:

Enter your answer here

2. **Subtraction**

- Yes
- No

Example of second number:

Enter your answer here

3. **Multiplication**

- Yes
- No

Example of second number:

Enter your answer here

Save Answer

Q3 Any Portability in a Storm

10 Points

Each subquestion should be considered in the context of appearing in `main` after the following definitions:

```
struct S {
   char c;
   int i;
};
```

```c
int iPlusOne(struct S s) {
   s.i++;
   return s.i;
}

int main(void) {
   char c;
   int i, j;
   unsigned int k;
   long l;
   struct S s = {'a', 2};
```

Q3.1

1 Point

This set of subquestions considers the following code:

```c
   i = iPlusOne(s) + iPlusOne(s);
```

Executing on `armlab`, what is the value of the `int` `i` after its assignment?

Enter your answer here

Save Answer

Q3.2

1 Point

Is this resulting value of `i` portable (i.e., would it be the same on any system running C90)?

- yes
- no

Save Answer

Q3.3

1 Point

Is the size of `i` portable (i.e., would it be the same on any system running C90)?

- yes
- no

Save Answer

Q3.4

1 Point

Is the size of `s` portable (i.e., would it be the same on any system running C90)?

- yes
- no

Save Answer

Q3.5

1 Point

This set of subquestions considers the following code:

```
c = '\0';
c--;
j = c;
```

Executing on `armlab`, what is the value of the `int` `j` after its assignment?

Enter your answer here

Save Answer

Q3.6

1 Point

Is this value of `j` portable to any system with character encodings using ASCII?

- yes
- no

Save Answer

Q3.7

1 Point

Is the size of the `char` `c` portable (i.e., would it be the same on any system running C90)?

- yes
- no

Save Answer

Q3.8

1 Point

This set of subquestions considers the following code:

```
k = 0U;
l = 1L;
if(k = l-1)
   k++;
else
   k--;
++k;
```

Executing on `armlab`, what is the value of the `unsigned int` `k` after this snippet?

Enter your answer here

Save Answer

Q3.9
1 Point

Is this resulting value of `k` portable (i.e., would it be the same on any system running C90)?

- O yes
- O no

Save Answer

Q3.10
1 Point

Is the size of the `long` `l` portable (i.e., would it be the same on any system running C90)?

- O yes
- O no

Save Answer

Q4 Places with Character (or not!)
20 Points

Consider the following C program, where some lines (`21-24`) are commented out initially.
Assume that all calls to `malloc` succeed (and you don't need to check this).

```
01 #include <stdio.h>
02 #include <stdlib.h>
03 #include <string.h>

04 int main(void) {
05   enum {ARRAY_LENGTH = 11};
06   int k = ARRAY_LENGTH / 2;
07   char place1[ARRAY_LENGTH] =
08     {'P', 'r', 'i', 'n', 'c', 'e', 't', 'o', 'n', '\0'};
09   char *place2 = "Kingston";
10   const char *place3 = "Queenston";
```

```
11   char *pp, *pq;

12   pp = (char *) malloc((ARRAY_LENGTH + 1) * sizeof(char));
13   strcpy(pp, place1);
14   pq = (char *) malloc((ARRAY_LENGTH + 1) * sizeof(char));
15   strcpy(pq, place3);
16   pp = pq;

17   if (place1[k] == pp[k])
18     printf("I love %s.\n", place1);
19   else
20     printf("I love %s.\n", place2);

21   /* free(pp);  */
22   /* pq = NULL; */
23   /* printf("Where is %s?!", pq);*/
24   /* free(place2); */

25   return 0;
26 }
```

As a reminder:

- For sizes of various `types` on `armlab`, see the relevant **handout** from precept 4.
- The library functions have the following calling conventions:

```
void *malloc(size_t size);
void free(void *ptr);
char *strcpy (char *destination, const char *source);
```

Q4.1

1 Point

How many bytes are allocated on the **stack frame** for the `main` function in this program? (Don't worry about alignment or padding.)

Enter your answer here

Save Answer

Q4.2

1 Point

How many bytes are allocated on the **heap** in this program? (Don't worry about alignment or padding, and assume that all calls to `malloc` succeed.)

Enter your answer here

Save Answer

Q4.3

1 Point

In which section of the memory is the variable `pp` stored **before** line `12`?

Enter your answer here

Save Answer

Q4.4

1 Point

In which section of the memory is the variable `pp` stored **after** line `12`?

Enter your answer here

Save Answer

Q4.5

6 Points

In which section(s) of the memory are the following strings stored:

A. `Princeton`

Enter your answer here

B. `Kingston`

Enter your answer here

C. `Queenston`

Enter your answer here

Save Answer

Q4.6

1 Point

What is the value of the variable `k` after line `06`?

Enter your answer here

Save Answer

Q4.7

1 Point

What is printed to `stdout` by lines `17-20` in the program ?

Enter your answer here

Save Answer

Q4.8

1 Point

How many bytes of **memory leak** does this program have (as shown)?

Enter your answer here

Save Answer

Q4.9
1 Point

Suppose we *uncomment* the code at line `21`.
How many bytes of memory leak does the program have now?

Enter your answer here

Save Answer

Q4.10
2 Points

Suppose we now uncomment the code at line `23` also, i.e., lines `21` and `23` are both uncommented.
Choose ***all*** the following options that apply.

- [ ] The program compiles with a warning.
- [ ] There is no bug and the program executes successfully (i.e., without failure).
- [ ] There is at least one bug, and the program definitely aborts.
- [ ] There is at least one bug, but the program may not abort.
- [ ] The program may print something to `stdout` at line `23`.
- [ ] The program may print `Where is Queenston?!` to `stdout` at line `23`.

Save Answer

Q4.11
2 Points

Suppose we now uncomment the code at line `22` also, i.e., lines `21–23` are uncommented.
Choose ***all*** the following options that apply.

- [ ] The program compiles with a warning.
- [ ] There is no bug and the program executes successfully (i.e., without failure).
- [ ] There is at least one bug, and the program definitely aborts.
- [ ] There is at least one bug, but the program may not abort.
- [ ] The program may print something to `stdout` at line `23`.
- [ ] The program may print `Where is Queenston?!` to `stdout` at line `23`.

Save Answer

Q4.12

2 Points

Finally, suppose we now uncomment the code at line `24` also, i.e., all lines `21-24` are uncommented.
Choose ***all*** the following options that apply.

- [ ] The program compiles with a warning.
- [ ] There is no bug and the program executes successfully (i.e., without failure).
- [ ] There is at least one bug, and the program definitely aborts.
- [ ] There is at least one bug, but the program may not abort.
- [ ] The program may print something to `stdout` at line `23`.
- [ ] The program may print `Where is Queenston?!` to `stdout` at line `23`.

Save Answer

Q5 No stresslessness for "possessionlessness"

5 Points

In each subquestion you will consider an implementation of the following function, which is another member of the `string.h` interface that you worked with in Assignment 2:

```
char* strrchr(const char* s, int c)
```

This function locates the last (i.e., highest address) occurrence of `c` (converted to a `char`) in the string pointed to by `s`. It returns a pointer to the located character or `NULL` if the character does not appear in the string. The terminating nullbyte is considered to be part of the string; therefore if `c` is `'\0'`, the function locates the terminating `'\0'`.

Q5.1

1 Point

```
01  char* strrchr1(const char* s, int c) {
02    char* ret = NULL;
03    assert(s != NULL);
04    while(*s) {
05      if(*s == c)
06        ret = s;
07      s++;
08    }
09    if(!c)
10      ret = s;
11    return ret;
12  }
```

When compiled with `gcc217`, `strrchr1`:

- compiles with no warnings or errors
- produces an error on line 4
- produces an error on line 9
- produces warnings on lines 6 and 10
- produces a warning on line 11
- more than one of the above

Save Answer

Q5.2

1 Point

```
01  char* strrchr2(const char s[], int c) {
02    size_t ret = 0;
03    size_t i = 0;
04    assert(s != NULL);
05    while(s[i] != '\0') {
06      if(s[i] == c)
07        ret = i;
08      i++;
09    }
10    if(c == '\0')
11      ret = i;
12    return (char*) &s[ret];
13  }
```

`strrchr2` produces the correct answer for:

- all possible strings `s` and characters `c`
- all possible strings `s` and characters `c`, but not efficiently (as defined in A2)
- some, but not all, possible strings `s` and characters `c`
- only the empty string `s`
- no strings `s`

Save Answer

Q5.3

1 Point

```
01  char* strrchr3(const char s[], int c) {
02    size_t i = 0;
03    assert(s != NULL);
04    while(s[i] != '\0') {
05      if(s[i] == c)
06        return (char*) &s[i];
07      i++;
08    }
09    if(c == '\0')
10      return (char*) &s[i];
11    return NULL;
12  }
```

`strrchr3` produces the correct answer for:

- ○ all possible strings `s` and characters `c`
- ○ all possible strings `s` and characters `c`, but not efficiently (as defined in A2)
- ○ some, but not all, possible strings `s` and characters `c`
- ○ only the empty string `s`
- ○ no strings `s`

Save Answer

Q5.4

1 Point

```
01  char* strrchr4(const char s[], int c) {
02    size_t i = strlen(s);
03    char* p = NULL;
04    for( ; i > 0; i--) {
05      if(s[i] == c)
06        return (char*) &s[i];
07    }
08    if(s[0] == c)
09      return (char*) &s[i];
10    return p;
11  }
```

`strrchr4` produces the correct answer for:

- all possible strings `s` and characters `c`
- all possible strings `s` and characters `c`, but not efficiently (as defined in A2)
- some, but not all, possible strings `s` and characters `c`
- only the empty string `s`
- no strings `s`

Save Answer

Q5.5

1 Point

```
01  char* strrchr5(const char* s, int c) {
02    const char* ret = NULL;
03    assert(s != NULL);
04    while(*s != '\0') {
05       if(*s == c)
06         ret = s;
07       s++;
08    }
09    if(c == '\0')
10      return (char*) s;
11    return (char*) ret;
12  }
```

`strrchr5` produces the correct answer for:

- all possible strings `s` and characters `c`
- all possible strings `s` and characters `c`, but not efficiently (as defined in A2)
- some, but not all, possible strings `s` and characters `c`
- only the empty string `s`
- no strings `s`

Save Answer

Q6 Sort out this mess!

8 Points

The following function `isSorted` should return `1` (true) if the argument array `a` of `n` integer elements is sorted in ascending order, i.e., its elements are in non-decreasing order, and `0` (false) otherwise.
However, it has several bugs.

```
1  int isSorted(int *a, size_t n){
2    size_t i;
3    int check;

4    /* add assert statement(s) here */

5    for (i=0; i < n; i++)
6       check = a[i] <= a[++i];
7    return check;
8  }
```

Q6.1

2 Points

To start with, add any missing ***assert*** statements at line `4`. You can assume that the header file `assert.h` has been included.

Enter your answer here

(Note: the missing assert(s) are not bugs *per se* -- adding them is recommended for testing code modularly.)

Save Answer

Q6.2

6 Points

Now identify up to ***three*** different bugs.
For each bug, describe the effect of the bug (concisely) and suggest a fix.

Assume that all required C header files are included, and don't worry about comments, efficiency, error checking, or style.

Bug 1:
Effect (concise explanation):

Enter your answer here

Suggested fix (show the fixed code, with line numbers):

Enter your answer here

Bug 2:
Effect (concise explanation):

Enter your answer here

Suggested fix (show the fixed code, with line numbers):

Enter your answer here

Bug 3:
Effect (concise explanation):

Enter your answer here

Suggested fix (show the fixed code, with line numbers):

Enter your answer here

THE END

Save Answer

Save All Answers

Submit & View Submission >