15-122: Principles of Imperative Computation, Fall 2017

Written Homework 12

Due: Monday 27$^{\text{th}}$ November, 2017 by 9pm

Name: Shaojie Zhang

Andrew ID: shaojiez

Section: O

This written homework provides practice with C features such as pointer arithmetic, undefined behaviors and casting.

Instructions

You can prepare your submission in one of two ways:

Just edit (preferred) Use any PDF editor (e.g., Preview on Mac, iAnnotate on mobile, Acrobat Pro installed on all non-CS cluster machines and most platforms) to typeset your answers in the given spaces — you can even draw pictures. *That's it.*

Print and Scan Alternatively, print this file, write your answers *neatly* by hand, and then scan it into a PDF file. *This is pretty labor-intensive.*

Once you have prepared your submission, submit it on Gradescope. You have unlimited submissions.

Question:	1	2	3	Total
Points:	3	3	6	12
Score:				

Evaluation Summary Once this homework is graded, you will be able to find a summary of your performance on Gradescope.

3pts

1. Pass by reference and arrays versus pointers in C

The following little program allocates and initializes an array of integers, then calls a function to swap two of its elements. Rewrite the function `main` in the box below to use array notation instead of pointer notation wherever possible.

```c
#include <stdlib.h>
#include <stdio.h>
#include "lib/xalloc.h"
#include "lib/contracts.h"

void swap(int *x, int *y) {
  REQUIRES(x != NULL && y != NULL);
  int t = *x;
  *x = *y;
  *y = t;
  return;
}

int main() {
  int* A = xmalloc(sizeof(int) * 10);
  for (int i = 0 ; i < 10 ; i++) {
    ASSERT(0 <= i);
    *(A + i) = i;
  }
  ASSERT(*(A+2) == 2);
  ASSERT(*(A+4) == 4);
  swap(A+2, A+4);
  ASSERT(*(A+2) == 4);
  ASSERT(*(A+4) == 2);

  printf("All tests passed.\n");
  return 0;
}
```

```
int main() {

      int  *A = xmalloc(sizeof(int)*10);
      for (int i=0; i<10; i++) {
        ASSERT(0<=i);
        A[i] = i;
        }
      ASSERT(A[2] == 2);
      ASSERT(A[4] == 4);
      swap(&A+2, &A+4);
      ASSERT(A[2] == 4);
      ASSERT(A[4] == 2);
      printf("All tests passed. \n");
      return 0;

}
```

2. C Program Behavior

Each of the following C programs contains one or more errors. *Briefly* explain what is conceptually wrong with each example. No credit will be given if you simply copy error messages from the compiler, the runtime system, or `valgrind`. Of course you are encouraged to use these tools to help you understand the problems.

0.5pts

2.1

```
#include <stdio.h>
#include <string.h>
int main() {
  char *w;
  strcpy(w,"C programming");      // copy string into w
  printf("%s\n", w);
  return 0;
}
```

We did not allocate enough space for the string so that we won't be able to execute strcpy.

0.5pts

2.2

```
#include <stdio.h>
#define MULT(X,Y) (X*Y)
int main() {
  int c = MULT(3+4,4+5);
  printf("(3+4)*(4+5) is = %d\n", c);
  return 0;
}
```

We won't get the correct answer that we want since the MULT function will give us 3 + 4*4 + 5 = 24 instead of 7*20 = 140.

0.5pts

2.3

```
#include <stdlib.h>
#include "lib/xalloc.h"

int main() {
  int* A = xmalloc(sizeof(int) * 12);
  int* B = A;
  for (int i = 0 ; i < 12 ; i++) {
    A[i] = i;
  }
  free(A);
  for (int i = 1 ; i < 12 ; i++) {
    B[i] = B[i] + B[i-1];
  }
  free(B);
  return 0;
}
```

We won't have the correct access to B when we freed A since we freed what B was pointing at.

0.5pts

2.4

```
#include <stdlib.h>
#include <stdio.h>
#include "lib/xalloc.h"

int main() {
  int* A = xmalloc(sizeof(int) * 32);
  for (int i = 0 ; i < 32 ; i++) {
    A[i] = i + 4;
  }
  int* B;
  for (B = A; *B != 0; B++) {
    printf("A[i]: %d\n", *B);
  }
  free(B);
  return 0;
}
```

The for loop is wrong since if no element in the array is zero, the loop will never end.

0.5pts

2.5

```
#include <stdio.h>
int main()
{
  int i = 0;
  int j = 0;
  for (i = 1; i <= 1000; i++);
    j = j + i;
  printf("The sum of the integers from 1 to 1000 inclusive = ");
  printf("%d\n", j);
  return 0;
}
```

In the for loop we are supposed to write {} instead of ;

0.5pts

2.6

```
#include <stdio.h>
int main() {
  printf("DAVE: Open the pod bay doors please, HAL\n");
  char∗ hal = "I’m sorry Dave, I’m afraid I can’t do that.";
  printf("HAL: %s\n", hal);
  if (∗hal = ’I’)
    printf("DAVE: Hello, HAL? Do you read me?\n");
  else
    printf("DAVE: What’s the problem?\n");
  return 0;
}
```

In the if statement, we should use the double equal sign.

3. **Integer Types**

2pts

3.1 Suppose that we are working with the usual 2's complement implementation of unsigned and signed **char** (8 bits, one byte), **short** (16 bits, two bytes) and **int** (32 bits, four bytes).

We begin with the following declarations:

```
short w = -15;
unsigned short x = 65524;
unsigned short y = 9;
int z = -65523;
```

Fill in the table below. In the third column, always use two hex digits to represent a **char**, four hex digits to represent a **short**, and eight hex digits to represent an **int**. You might find these numbers useful: $2^8 = 256$, $2^{16} = 65536$ and $2^{32} = 4294967296$.

Most, but not all, of these answers can be derived from the lecture notes. If you can't find an answer from the lecture notes, you can look at online C references or just compile some code.

C expression	**Decimal value**	**Hexadecimal**
`w`	-15	0xFFF1
`(unsigned short) w`	65521	0xFFF1
`(int) w`	-15	0xFFFFFFF1
`x`	65524	0xFFF4
`(int) x`	65524	0xFFFFFFF4
`(int)(short) x`	-12	0xFFFFFFF4
`y`	9	0x0009
`(int)(short)y`	9	0x00000009
`z`	-65523	0xFFFF000D
`(unsigned int) z`	4294901773	0xFFFF000D

3pts

3.2 For this question, assume that **`char`** is a 1-byte signed integer type and that **`unsigned int`** is a 4-byte unsigned integer type.

Write the C function `pack_cui` which takes a **`char`** array of length 4 and packs it into a single **`unsigned int`**. We want the 0th character aligned at the most significant byte, and the last character aligned at the least significant byte. For example, given an array `C = {1, 2, -1, 4}`, `pack_cui(C)` should return `0x0102FF04`.

For full credit,

- Make all casts explicit.
- Do not cast (or otherwise convert types) directly between signed and unsigned types of different sizes.
- Do not rely on the *endianness*[1] of your machine. For example, the following code is incorrect:

  ```
  unsigned int pack_cui(char* C) { return *((unsigned int*) C); }
  ```

- Make sure your solution works for **`char`** arrays containing negative values.
- Write code that is clear and straightforward.

```
unsigned int pack_cui(char *C) {
        uint32_t u1 = C[0] << 24 && 0xFFFFFFFF;
        uint32_t u2 = C[1] << 16 && 0xFFFFFF00;
        uint32_t u3 = C[2] << 8 && 0xFFFF0000;
        uint32_t u4 = C[3] && 0xFF000000;
        unsigned int res;
        res = u1 | u2 | u3 | u4;
        return res;
}
```

[1] "Endianness" refers to the natural storage order of bytes for a particular hardware architecture; you can read about it on Wikipedia, and don't forget to read *Gulliver's Travels* in your no doubt copious spare time.

1pt

3.3 Suppose we've defined the following functions:

```
int fib(int n);  // returns the nth fibonacci number
int cat(int n);  // returns the nth catalan number
int las(int n);  // returns the nth look-and-say number
```

Complete the code below such that it will print

```
0 1 1 2 3 5
1 1 2 5 14 42
1 11 21 1211 111221 312211
```

(*Hint:* The **typedef** on the first line should define the type `int2int_fn`. This type should match the type of a function such as `fib`, `cat`, or `las`.)

```
typedef ___________int int2int_fn(int n)_____________________________;

void map_print(int2int_fn* f, int* A, size_t n) {
  for (size_t i = 0; i < n; i++) {

    int x = _____(*f)(A[i])____________________________________;
    printf("%d ", x);
  }
  printf("\n");
}

int main() {
  int A[6] = {0, 1, 2, 3, 4, 5};

  map_print(_________&fib________________________________, A, 6);

  map_print(_________&cat________________________________, A, 6);

  map_print(_________&las________________________________, A, 6);
  return 0;
}
```