CS/ECE 354 Spring 2016 - Midterm Exam Solutions

Part I - C Programming

C Basics

1. [2 points] In C, arguments to a function are always
   a. **Passed by value (i.e. the called function is given the values of its arguments in temporary variables rather than the originals)**
   b. Passed by reference (i.e. the called function has access to the original argument, not a local copy)
   c. Non-pointer variables are passed by value and pointers are passed by reference

2. [3 points] What is the problem with the following code and how will you correct it?

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

int main()
{
    int *p = malloc(sizeof(int));
    *p = 42;
    p = malloc(sizeof(int));
    free(p);
}
```

   ANSWER:

Pointers and Functions

3. [5 points] Does the following code run successfully to return 0 or does it generate a segmentation fault? If it runs fine, then what is the output? Otherwise explain why it segfaults.

A **segmentation fault** occurs when a program attempts to access a memory location that it is not allowed to access.

```c
#include <stdio.h>
#include <stdlib.h>

void populate(int *a)
{
    int *parray = malloc(2 * sizeof(int));
    parray[0] = 37;
    parray[1] = 73;
    a = parray;
}

int main()
{
    int *a = NULL;
    populate(a);
    printf("a[0] = %d and a[1] = %d\n", a[0], a[1]);
    return 0;
}
```

ANSWER:

Predict the output

Consider the following two programs, both of which are compiled and run on a CSL lab machine using the -m32 option.

```
sizeof(int) = 4 bytes
sizeof(char *) = 4 bytes
sizeof(char) = 1 byte
```

4. [3 points] What is the output of the code below on a 32-bit **little endian** machine?

    Note: The %x format specifier in printf prints the contents in **hexadecimal** notation.

```
#include <stdio.h>
int main() {
    int number = 288;
    char *ptr;
    ptr = (char *) &number;
    printf("%x",*ptr);
    return 0;
}
```


5. [4 points] What is the output of the code below?

```
#include <stdio.h>
int main() {
    char string[] = "BADGERS";
    char *ptr = string;
    *ptr = *ptr + 2;
    ptr = ptr + 2;
    printf("%c", *ptr);
    ptr--;
    printf("%c", *ptr);
    ptr = string;
    printf("%c", *ptr);
    return 0;
}
```


Linked Lists

6. [2 points] Assuming the following line of code is inside the main( ) function, in which part of memory is the pointer variable **parray** allocated and in which part of memory is the 10 element integer array allocated?

```
int *parray = malloc (sizeof (int) * 10);
```

The 4 types of program memory are: Code, Data, Stack and Heap.

10 element integer array is allocated in **heap** memory.

7. [5 points] What does the following `mystery_function()` do?

```
struct node {
    int data;
    struct node *next;
}

void mystery_function(struct node *head)
{
    struct node *temp = head;
    struct node *prev;

    if (temp == NULL) {
        printf("Linked List is empty.\n");
    } else {
        prev = temp;
        while (temp != NULL) {
            temp = temp->next;
            free(prev);
            prev = temp;
        }
    }
}
```
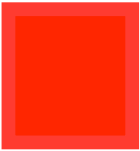
Structures

8. [6 points] What is the size (in bytes) for the following structures on a 32-bit machine installed with the Linux Operating System? The first one is already answered for you! :)

$$sizeof(int) = 4 \text{ bytes}$$
$$sizeof(short) = 2 \text{ bytes}$$
$$sizeof(char) = 1 \text{ byte}$$

Structure	Size (in bytes)
```struct foo { int d1; char c1; int d2; } ```	
```struct foo { int d1; char c1; int d2; char c2; short s; }; ```	
```struct foo { int d1; int d2; char c1; char c2; short s; }; ```	
```struct foo { char c1; int d1; short s; int d2; char c2; }; ```	

Part II - Data Representation

9. [12 points] Suppose that x and y have byte values 0x93 and 0x3F, respectively. Fill in the following table indicating the byte values of the different C expressions. All values must be written in hexadecimal notation. The first one is already answered for you! :)

Expression	Value
x & y	
~x | ~y	
x & !y	
!x || !y	
x && ~y	
x << 3	
x >> 2 (arithmetic)	

10. [8 points] The following function has a bug and doesn't work as expected. What is the issue with this function and how will you fix it?

```
// If x is greater than y, this function should return 1.
// Else, this function returns 0.
int is_greater(unsigned int x, unsigned int y)
{
    if (x - y > 0)
        return 1;
    else
        return 0;
}
```

ANSWER:

Part III - Assembly Programming

Addressing

11. [4 points] Assume the following memory layout, with the following values for the registers:

$$\%edx = 0x8049000$$
$$\%ecx = 0x5$$

Memory Address	Value in Memory
	...
0x8049024	0x8049028
0x8049020	0x8049024
0x804901c	0x8049020
0x8049018	0x804901c
0x8049014	0x8049018
0x8049010	0x8049014
0x804900c	0x8049010
0x8049008	0x804900c
0x8049004	0x8049008
0x8049000	0x8049004
	...

What is the value in the register %eax after each of the following assembly instructions? The first one is already answered for you! :)

Assembly Instruction	Value in register %eax
movl $0x8049000, %eax	

movl 0x8049000, %eax	
movl (%edx), %eax	
leal 4(%edx), %eax	
movl 4(%edx, %ecx, 4), %eax	

Assembly to C

12. [6 points] Assume variables a and b are stored at -0x8(%ebp) and -0x4(%ebp) respectively. Write the equivalent C expressions for the following assembly snippets. The first one is already answered for you! :)

S.No.	Assembly Instruction	Corresponding C code

1	`movl    -0x4(%ebp),%eax`
	`movl    (%eax),%eax`
	`movl    %eax,-0x8(%ebp)`	

2	`leal    -0x4(%ebp),%eax` `movl    %eax,-0x8(%ebp)`	
3	`movl    -0x4(%ebp),%eax` `movl    %eax,-0x8(%ebp)`	

4	`movl    -0x4(%ebp),%eax`
	`movl    (%eax),%edx`
	`movl    -0x8(%ebp),%eax`
	`movl    %edx,(%eax)`	

5	`movl    -0x4(%ebp), %eax`
	`movl    (%eax), %edx`
	`movl    -0x8(%ebp), %eax`
	`movl    (%eax), %eax`
	`addl    %eax, %edx`
	`movl    -0x4(%ebp), %eax`
	`movl    %edx, (%eax)`	

Arrays and Structures

13. [7 points] Assume variable **a** is stored starting at memory address **0x8049000.** What is the value of the memory location stored in the pointer variable p for the following cases? Write the value of p in hexadecimal notation. The first one is already answered for you!

sizeof(int) = 4 bytes

S.No.	C code	Hex value in pointer p
1	`int a[10];` `int *p = &a[7];`	

2	`struct foo {`
	`    int x;`
	`    int y;`
	`    int z;`
	`};`
	
	`struct foo a;`
	`int *p = &a.y;`	

3	`struct foo {`
	`    int x[4];`
	`    int y[4];`
	`};`
	
	`struct foo a;`
	`int *p = &a.y[2];`	

4	`struct foo {`
	`    int x[4];`
	`    int y[4];`
	`};`
	
	`struct foo a[10];`
	`int *p = &a[7].y[2];`	

Control Flags

14. [9 points] The function fun( ) takes three integer arguments x, y, and z and returns a character as shown below:

$$\text{char fun(int x, int y, int z);}$$

The arguments x, y, and z are stored at memory address 0x8(%ebp), 0xC(%ebp), and 0x10(%ebp) respectively.

For the various assembly programs shown below, you are expected to fill in the correct data type cast (e.g. char, short, int) within the parenthesis ( ) and the correct comparison operators(e.g. <, <=, >, >=, !=) in the blank space provided between the two operands. The first one is already answered for you! :)

The following are the data types and the comparison operators that you are allowed to use in the questions below:
Allowed data types: `int, char, short, and unsigned`
Allowed comparison operators: `>, <, >=, <=, !=, ==`

movsbl S,D : Move sign-extended byte to double word (D ← SignExtend(S))

movsbw S,D : Move sign-extended byte to word (D ← SignExtend(S))

cmpw S2,S1 : Compare word (based on S1-S2)

For the code snippets 1, 2, 3, and 4 (shown below), the char variable `t` is present at memory locations `-0x4(%ebp), -0x6(%ebp), -0x5(%ebp), and -0x3(%ebp)` respectively.

S.No.	Assembly Instruction	Corresponding C code

1	`movl    0x8(%ebp),%eax`
	`movsbl  %al,%edx`
	`movl    0xc(%ebp),%eax`
	`cmpl    %eax,%edx`
	`seta    %al`
	`movb    %al,-0x4(%ebp)`	

2	`movl    0x10(%ebp),%edx`
	`movl    0x8(%ebp),%eax`
	`cmpl    %eax,%edx`
	`setbe   %al`
	`movb    %al,-0x6(%ebp)`	

3	`movl    0x10(%ebp),%eax`
	`cmpl    0x8(%ebp),%eax`
	`setge   %al`
	`movb    %al,-0x5(%ebp)`	

4	`movl    0x8(%ebp),%eax`
	`movsbw  %al,%dx`
	`movl    0x10(%ebp),%eax`
	`cmpw    %ax,%dx`
	`setne   %al`
	`movb    %al,-0x3(%ebp)`	

Jumps

15. [4 points] In the disassembled version of the machine code shown below, instructions at addresses 0x66, 0x6c, and 0x73 use relative encoding to encode jump targets. All the values in the disassembled code below (including the address column) are represented using hexadecimal numbers.

```
Address    Opcode                  Assembly Code
0x62:      83 7d 08 00             cmpl    $0x0,0x8(%ebp)
0x66:      74 0d                   je      X
0x68:      83 7d 0c 00             cmpl    $0x0,0xc(%ebp)
0x6c:      74 07                   je      Y
0x6e:      b8 01 00 00 00          mov     $0x1,%eax
0x73:      eb 05                   jmp     Z
0x75:      b8 00 00 00 00          mov     $0x0,%eax
0x7a:      85 c0                   test    %eax,%eax
0x7c:      0f 9e c0                setle   %al
0x7f:      88 45 ff                mov     %al,-0x1(%ebp)
```

What is the value of the addresses X, Y and Z? The values of X, Y, and Z should be written in hexadecimal notation. The first one is already answered for you! :) Remember

11

the addresses X, Y, and Z, in assembly code will refer to actual addresses (e.g. 0x75) even though relative encoding is used in the machine code (opcode).

ANSWER:

Loops

16. [6 points] An assembly code and its corresponding C code is given below. Fill in the parts in the C code that are missing.

Assembly Code	C Code

<pre>loop_func:
    pushl   %ebp
    movl   %esp, %ebp
    subl $16, %esp
    movl $0, -4(%ebp)
    jmp .L2
.L3:
    movl   8(%ebp), %eax
    addl   %eax, -4(%ebp)
    subl $1, 8(%ebp)
.L2:
    cmpl $0, 8(%ebp)
    jg   .L3
    movl   -4(%ebp), %eax
    leave
    ret</pre>
	<pre>int loop_func(int n)
	{
	    int sum = 0;
	
	    while (▮) {
	
	       ▮
	
	    }
	
	    return ▮
	}</pre>

Functions

17. [8 points] Match the following C functions (C1, C2, C3 and C4) with their corresponding assembly functions (A1, A2, A3, and A4). Write your answers in the spaces provided at the end of the questions.

12

| C program	Assembly Program

C1
 	
``c
int func(int x, int y)
{
    int result = x && y;
    return result;
}
``
	A1
	
	``
	func:
	  pushl %ebp
	  movl %esp, %ebp
	  subl $16, %esp
	  movl 8(%ebp), %eax
	  orl 12(%ebp), %eax
	  movl %eax, -4(%ebp)
	  movl -4(%ebp), %eax
	  leave
	  ret
	``

C2
 	
``c
int func(int x, int y)
{
    int result = x || y;
    return result;
}
``
	A2
	
	``
	func:
	  pushl %ebp
	  movl %esp, %ebp
	  subl $16, %esp
	  cmpl $0, 8(%ebp)
	  je .L2
	  cmpl $0, 12(%ebp)
	  je .L2
	  movl $1, %eax
	  jmp .L3
	.L2:
	  movl $0, %eax
	.L3:
	  movl %eax, -4(%ebp)
	  movl -4(%ebp), %eax
	  leave
	  ret
	``

C3	A3
``int func(int x, int y)
{
int result = x & y;
return result;
}```	```func:
pushl %ebp
movl %esp, %ebp
subl $16, %esp
movl 8(%ebp), %eax
andl 12(%ebp), %eax
movl %eax, -4(%ebp)
movl -4(%ebp), %eax
leave
ret```
C4	**A4**
``int func(int x, int y)
{
int result = x	y;
return result;
}```	```func:
pushl %ebp
movl %esp, %ebp
subl $16, %esp
cmpl $0, 8(%ebp)
jne .L2
cmpl $0, 12(%ebp)
je .L3
.L2:
movl $1, %eax
jmp .L4
.L3:
movl $0, %eax
.L4:
movl %eax, -4(%ebp)
movl -4(%ebp), %eax
leave
ret```

Write your answers below: (If C1 matches with A4, write A4 in the space next to C1)
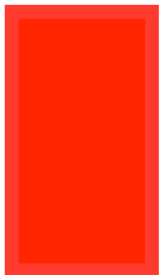
Recursion (Dream within a dream!)

18. [6 points] Consider the following recursive factorial function (shown in the lecture) in C and Assembly language. The line numbers for rfact in assembly are in decimal.

```c
int rfact(int n)
{
    int result;
    if (n <= 1)
        result = 1;
    else
        result = n * rfact(n-1);
    return result;
}
```

Line# Assembly Code
```
 1.   rfact:
 2.        pushl %ebp
 3.        movl %esp, %ebp
 4.        pushl %ebx
 5.        subl $4, %esp
 6.        movl 8(%ebp), %ebx
 7.        movl $1, %eax
 8.        cmpl $1, %ebx
 9.        jle .L53
10.        leal -1(%ebx), %eax
11.        movl %eax, (%esp)
12.        call rfact
13.        imull %ebx, %eax
14.   .L53:
15.        addl $4, %esp
16.        popl %ebx
17.        popl %ebp
18.        ret
```

Questions:

1. Why do we push the %ebx register's value on the stack frame of rfact?
    (Refer: Line# 4 in assembly code - `pushl %ebx`)

2. What is the purpose of the following 2 statements?
    a. `subl $4, %esp (Line number 5)`

        recursive call of rfact.

    b. `addl $4, %esp (Line number 15)`

3. For every invocation of the function rfact( ) which register is used to store the value of its input argument?

    %ebx

4. What is the purpose of the following line of assembly code (Line number 10)?

    `leal -1(%ebx), %eax`

5. Why are the following 2 lines (Line numbers 2 - 3) needed in `rfact()` function?
    ```
    pushl %ebp
    movl %esp, %ebp
    ```