Midterm - KEY - Maximum 84 Points

CSE 109: Systems Programming (Fall 2018)

October 10th, 2018

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Lehigh Email ID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Instructions: Write down your name and Lehigh Email in the above spaces; you have 75 minutes; this is a closed-book, closed-notes exam; all calculators, PDAs, portable audio players and cell phones must be put away for the duration of the exam. This exam is single-sided, no work/answers on the backs of any pages will be accepted.*

**Write down your Lehigh Email at the top of each following page, do not put your name on those pages. Do that now. Seriously, don't screw that up. Your pages may not get graded if you do not do this.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| int strcmp(char \*s1, char \*s2) | Compares s1 and s2, returns difference |  | DEC | HEX | BIN |
| char \*strcpy(char \*dest, char \*src) | Copies src into dest, returns dest |  | 0 | 0 | 0000 |
| char \*strcat(char \*dest, char \*src) | Appends src onto the end of dest; returns dest |  | 1 | 1 | 0001 |
| int atoi(char \*nptr) | Returns integer representation of nptr |  | 2 | 2 | 0010 |
| int atoi(char \*nptr) | Returns integer representation of nptr |  | 3 | 3 | 0011 |
| int isupper(int x) | Returns 1 if x is uppercase, else 0 |  | 4 | 4 | 0100 |
| int isdigit(int x) | Returns 1 if x is a digit, else 0 |  | 5 | 5 | 0101 |
| int isspace(int x) | Return 1 if x is a space, else 0 |  | 6 | 6 | 0110 |
| int | 4 bytes |  | 7 | 7 | 0111 |
| pointers | 8 bytes |  | 8 | 8 | 1000 |
| char | 1 byte |  | 9 | 9 | 1001 |
| short | 2 bytes |  | 10 | A | 1010 |
| float | 4 bytes |  | 11 | B | 1011 |
| double | 8 bytes |  | 12 | C | 1100 |
| & | Bitwise AND |  | 13 | D | 1101 |
| && | Logical AND |  | 14 | E | 1110 |
| | | Bitwise OR |  | 15 | F | 1111 |
| || | Logical OR |  |  |  |  |
| ^ | Exclusive OR (Symmetric Difference) |  |  |  |  |

1. (8 points) What will appear on the screen and/or in the log.txt file after compiling to produce a prog executable?

prog > log.txt

#include<stdio.h>

void changeX(int);

int main() {

int x = 10;

fprintf(stderr, "Before function call\n");

fprintf(stdout, "x = %d\n", x);

changeX(x);

fprintf(stdout, "After function call\n");

fprintf(stderr, "x = %d\n", x);

return 0;

}

void changeX(int x) {

x -= 1;

x = 1 - x;

--x;

}

On screen (if anything) In log.txt (if anything)

Before function call (2 pts) x = 10 (2 pts)

x = 10 (2 pts) After function call (2 pts)

*Subtract points for extra stuff in either file*

2. (6 points) Describe a modular approach to code development? List 3 reasons (2 points each) why writing modular code is beneficial/desirable?

Multiple files with each one having a singular or very limited functionality/purpose

Allows simultaneous code development by multiple coders

Simplifies development

Simplifies testing

Simplifies debugging

Allows for easy reuse

*Any other reasonable point*

3. (15 points) Pointers: Consider the following memory layout, as per Pointer Lab. Answer the following questions. Note that when we indicate an array type, we are stating where its elements begin, for everything else, we are indicating where that variable's data is located in memory.

Data set starting at 0xdb1100 of size 128.

0xdb1100: 50 11 db 00 00 00 00 00 40 11 db 00 00 00 00 00 P.......@.......

0xdb1110: 30 11 db 00 00 00 00 00 44 11 db 00 00 00 00 00 0.......D.......

0xdb1120: 20 11 db 00 00 00 00 00 24 11 db 00 00 00 00 00 .......$.......

0xdb1130: 58 11 db 00 00 00 00 00 70 11 db 00 00 00 00 00 X.......p.......

0xdb1140: 00 11 db 00 00 00 00 00 28 11 db 00 00 00 00 00 ........(.......

0xdb1150: 08 11 db 00 00 00 00 00 0c 11 db 00 00 00 00 00 ................

0xdb1160: 04 11 db 00 00 00 00 00 1c 11 db 00 00 00 00 00 ................

0xdb1170: 38 11 db 00 00 00 00 00 38 11 db 00 00 00 00 00 8.......8.......

int array[] is stored at: 0xdb1108

int \*ptr is stored at: 0xdb1160

short \*shortPtr is stored at: 0xdb1130

int x is stored at: 0xdb1138

short y is stored at: 0xdb1112

int \*\*\*what is defined by: (int \*\*\*)(((int \*\*)0xdb1118) + 1)

(a) (2.5 points) What is the value of x? *see Loew’s key 0xdb1170*

(b) (2.5 points) What is the value of y? *see Loew’s key 0xdb*

(c) (2.5 points) What is shortPtr[4]? *see Loew’s key 0x1104*

(d) (2.5 points) What is &ptr[3]? *see Loew’s key &\*(ptr + 3) 0xdb1110*

(e) (2.5 points) What is \*\*(int \*\*)(array + 2)? *see Loew’s key 0xdb1158*

(f) (2.5 points) What is \*\*\*what? *see Loew’s key 0xdb1120*

4. (4 points) Is C a pass-by-value or pass-by-reference language? Circle your answer.

Pass-by-value Pass-by-reference

5. (4 points) What is the difference between a procedural language and an object-oriented language?  
  
A procedural language uses a series of well-structured steps; i.e., a systematic order of statements, functions and commands. An object-oriented language focuses on objects which may encapsulate both data and functions to simplify development and subsequent code changes.

6. (4 points) Answer the following questions about data in memory.

(a) (2 points) What is meant by the “endianness” of a machine/system?

Endianness refers to the order in which multi-bytes objects are stored in memory. If the machine is “Big Endian”, then the MSB is stored at the start of an address. If the machine is “Little Endian”, then the LSB is stored at the start of an address.

(b) (2 points) Why is it important for a programmer to know this?

Endianness is important for a programmer to know because the endianness may be different between the system for which a programmer is writing a network application and the network itself; and if not accounted for, errors can result.

OR

Endianness is important for a programmer to know because the programmer may be accessing a multi-byte object in memory a byte at a time (perhaps through a union or some other mechanism).

OR

When looking at memory, the bytes in a multi-byte objects will appear reversed.

7. (8 points) Answer the following questions regarding the virtual memory available to each process.  
  
(a) (2 points) In what section/area/region in memory are temporary/automatic variables allocated?  
  
 stack  
  
(b) (2 points) Why would a programmer ever need to request memory be allocated?  
  
 Memory amount needed is not known prior to compilation   
 E.g., new records being added by the user during program execution. (*example not necessary*)

(c) (2 points) In what section/area/region will this requested memory be allocated?

heap

(d) (2 points) Will variables allocated here be global?  
  
yes

8. (4 points) Answer the following regarding make.

(a) (2 points) What is the purpose of a makefile (and the make command)?

With medium to large software projects containing many files, it’s difficult to type commands to compile all the files correctly each time; a makefile (when run by make) keeps track of which files have been changed, and each files’ dependencies on other files. (*any similar response is acceptable*)

(b) (2 points) List 2 benefits.

Prevents error

Faster compiles when only a subset of files changed

Convenient

*Any other reasonable response is OK.*

9. (4 points) Answer the following.

1. What is the difference between a C++ class and a C structure? (2 points)

A class may also include methods/functions which operate on the class’ data. This is encapsulation.

1. What is the difference between an object and an instance? (2 points)

There is no difference.

10. (6 points) Answer the following questions.

1. (3 points) What does the compilation stage of the compiler perform?

~~Takes the assembler source code and produces an assembly listing with offsets.~~ Takes output of the preprocessor and the source code, and generates assembler source code.

1. (3 points) Is the output unique to the microarchitecture of the target system?

Yes, the assembly code is unique to the instruction set architecture and optimized for the microarchitecture of the target system.

11. (2 points) What is a pointer?

A pointer is a variable containing an address to an object matching the type of the pointer.

12. (10 points) Answer the following questions about the structure provided below:

**struct** test\_first

{

**double** a;

**unsigned int** b;

**struct** test\_second

{

**float** a;

**unsigned char** b;

**float** c;

**unsigned char** d[6];

**char** e;

**unsigned char** f;

**int** g;

**struct** test\_third

{

**unsigned** char a;

**unsigned** char b;

**long long** c;

} h;

} c;

};

(a) What is the size (1.5 points) and alignment (1 point) of struct test\_first? *see Loew’s key*

(b) What is the size (1.5 points) and alignment (1 point) of struct test\_second? *see Loew’s key*

(c) What is the size (1.5 points) and alignment (1 point) of struct test\_third? *see Loew’s key*

(d) If we could rearrange any of the declarations of the elements contained within any of these structures, what is the minimum size the resulting struct test\_first could have (2points)? Write out the resulting structure (0.5 points).

*see Loew’s key*

13. (3 points) What does the following statement mean: Arrays in C are non-assignable and non-copy-initializable? (Aside from defining during declaration.) What is used instead?

One cannot simply assign a string constant to an array nor can one copy one array to another array using the assignment operator (“=”). Instead, the programmer should use one of the existing string functions, like strcpy, or something similar of their own construction.

14. (6 points) Answer the following questions regarding member access.

1. (4 points) What are the two member access operators associated with both C structures and C++ classes and when is each used?

The “.” and “->” symbols serve as member access operators. The “.” operator is used when accessing the structure or class directly (i.e., not through a pointer). The “->” operator is used when accessing the structure or class via a pointer.

1. (2 points) How does a class access its own member functions?

An object can access its own member functions by using the “this” pointer, an implicit pointer to the address of the object itself.