DALHOUSIE UNIVERSITY
Inspiring Minds

CSCI 2132
Final Exam Solutions

Term: Fall 2018 (Sep4-Dec4)

1. (12 points) True-false questions. 2 points each. No justification necessary, but it may be helpful if the question seems ambiguous.

a) (2 points) In the UNIX file model, a file is a stream of bytes.

> **Solution:** True.

b) (2 points) Symbolic links and pipes are types of files in Unix.

> **Solution:** True. (Symbolic links and pipes are types of files in Unix. The seven types of files in Unix are: 1. regular files, 2. directory files, 3. buffered special files, 4. unbuffered special files, 5. symbolic links, 6. pipes, and 7. sockets.)

c) (2 points) In the C programming language, the following two pairs of `scanf` format strings are equivalent:
`"%c,%c"`
`"%c ,%c"`

> **Solution:** False.

d) (2 points) In the C programming language, the following two pairs of `scanf` format strings are equivalent:
`"%d,%d"`
`"%d ,%d"`

> **Solution: False.**

e) (2 points) If there is a text file named `log.txt` in the current working directory, the command `date >> log.txt` will append the output of the `date` command to the end of `log.txt`.

> **Solution:** True.

f) (2 points) In a shell script, whenever we used the command `exit` to terminate the script, we are always required to provide an exit code explicitly as the argument of the `exit` command.

> **Solution:** False

2. (12 points) Multiple-choice. Circle the *single* best answer.

a) (3 points) Which of the following statements regarding streams and files in the C programming language is INCORRECT?

A. In C, a stream is any source of input or any destination of output;

B. The following statement will output "out of memory" followed by a newline character to the standard error channel:
`fprintf(stdout, "out of memory\n");`

C. The `fopen` function returns NULL when it fails to open a file;

D. The notion of lines does NOT apply to binary files.

Solution: B.

---

b) (3 points) If `c` is a variable of type `char`, which one of the following statements is illegal?

A. `i += c;  /* i has type int */`

B. `c = 2 * c - 1;`

C. `printf(c);`

D. `putchar(c);`

Solution: C.

---

c) (3 points) Suppose that the following declarations are in effect:

```
int a[] = {17, 211, 10, -5, 14, 19};
int *p = &a[3];
```

What is the value of `*(p+2)`?

A. 10

B. -5

C. 14

D. 19

Solution: D.

---

d) (3 points) A file `names` contains people names. Each line contains a first name and a last name of a person (e.g. `John Smith`), and the file is not sorted. Our task is to list only first names, sorted alphabetically, and no name should be repeated in more than one line.

Which one of the following four command will complete this task correctly?

A. `cut -d " " -f 1 < names | uniq | sort`

B. `cat names | cut -d " " -f 1 | sort | uniq`

C. `sort | uniq | cut -d " " -f 1 names`

D. `sort < names | uniq | cut -d " " -f 1`

Solution: **B.**

3. (6 points) Program Output.

What is the output of the following program? You can justify your answer, but it is not required.

```
int a[10] = {1}, i;
int *p = &a[4];
int *q = p+1;

for (i=1; i<10; i++) a[i] += a[i-1]+2;

*(p++) = *(q++); q++; *(p++) = *(q++);

printf(" %d %d %d\n", *a + *q - *p, q-p, *q-*p);

for (i = 4; i < 8; i++) printf(" %d", a[i]);
```

Solution: Short answer:

```
 5 2 4
 11 15 13 15
```

[Marking scheme: 3 points each row]

After the first three lines, which start with 'int', we have:

```
   0 1 2 3 4 5 6 7 8 9
a: 1 0 0 0 0 0 0 0 0 0
           p q
```

After the first for-loop:

```
   0 1 2 3 4  5  6  7  8  9
a: 1 3 5 7 9 11 13 15 17 19
           p q
```

After the next line with pointer arithmetic:

```
   0 1 2 3  4  5  6  7  8  9
a: 1 3 5 7 11 15 13 15 17 19
                   p     q
```

Now, the first printf command will produce:

```
5 2 4
```

because: $1 + 17 - 13 = 5$, $8 - 6 = 2$, and $17 - 13 = 4$.
The next for-loop prints the value of array `a` from 4 to 7, which are:

```
(i=4)11 (i=5)15 (i=6)13 (i=7)15
```

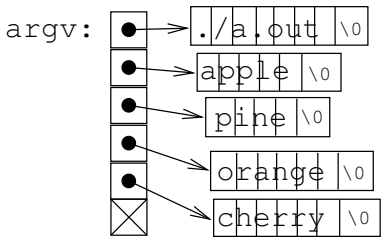
4. (6 points) Explain briefly.

Show the content of the standard `main` parameters: `argc` and `argv` when we call the compiled program in the following way:
`./a.out apple pine orange cherry`
If there are any pointers or strings, show them graphically.

Solution: The parameter `argc` contains the total number of arguments in the command line, including the program name, so its value is:

`argc: 5`

The parameter `argv` is an array of pointers to the arguments, and its value can be shown graphically as follows:



No marks are lost if the final NULL pointer in `argv` is not shown.

Marking scheme: 2pt for argc, 4pt for argv.

5. (7 points) Program Output. What is the output of the following program?

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

int main() {
  char s[90] = "test";
  char *p = "program";

  printf("%d %d\n", strlen(s), strlen(p+2));
  puts(strcat(s, p));
  printf("%d\n", strlen(s));

  return 0;
}
```

Solution:

```
4 5
testprogram
11
```

6. (9 points) Single command line. For each of the following questions, write a single Unix command line to perform the task required.

a) (3 points) Print a list of files in the directory `/usr/bin` whose names end with the English word "make". The file named `make` is considered one of these files.

Solution:

```
ls /usr/bin/*make
```

b) (3 points) Print out the number of six-character words in the Linux dictionary file `/usr/share/dict/linux.words` starting with `a` or `c` and ending with `h`.

Recall that this dictionary file contains one English word per line.

Solution:

```
grep '^[ac]....h$' /usr/share/dict/linux.words | wc -l
```

c) (3 points) Print out a list of regular files in the directory `/usr/bin` whose file owner has read permission but does not have execute permission.

Solution:

```
ls -l  /usr/bin | grep "^-r.-"
```

7. (8 points) Give concise answers.

a) (4 points) Write down the full names (NOT abbreviations) of four `gdb` commands, i.e. those commands that you can enter in the `gdb` console. For each command you put down, write a short sentence to explain what this command is for. You are not required to explain the usage such as parameters or options of these commands; the explanation that you write just have to be sufficient to show that you know what these commands do.

Solution:

```
E.g.,
next -- executes next line of code
step -- executes the next line of code, but steps into function when a
        function call is executed
continue -- continues execution until the next breakpoint or program
       termination
breakpoint line -- to set a breakpoint
```

b) (4 points) Suppose that you are working in a directory that is a working copy of a directory in SVN. You created a new file, named `file.c` in your directory. Which command lines are required to save this file in the SVN repository?

Solution:

```
svn add file.c
svn commit -m'file.c added'
```

8. (8 points) Give concise answers.

a) (4 points) Briefly explain the keyword `static` when used with a variable inside a function and outside a function.

> **Solution:** When the keyword `static` is used inside a function, the declared variable is a static variable that keeps its value from function call to function call, but it is visible only inside the function.
>
> When the keyword `static` is used outside of a function, the declared variable is static but visible only inside the current source code file, and not from other source code files.

b) (4 points) The following is supposed to be an implementation of the `strcpy` function in the standard C library, but there are two errors. Find out these errors, explain why they are incorrect, and fix these errors by modifying the code printed below.

```
char* strcpy(char *s1, const char *s2) {
  int i = 0;

  while (s2[i] != '\0') {
    s1[i] = s2[i];
    i++;
  }

  return s2;
}
```

> **Solution:** Error 1: `s1` will not be NULL-terminated after the function call. This is an error as C strings are terminated with NULL. Add `s1[i] = '\0';` after the loop.
>
> Error 2: The function returns `s2` instead of `s1`. Change `return s2;` to `return s1;`

9. (10 points) Large program organization.

(10 points) Assume that you have a project with the following files: `main.c` containing the main function, `llist.c` containing a linked list implementation, and `llist.h` a header file containing all function prototypes. The file `llist.h` is included in the two source files `main.c` and `llist.c`.

a) (4 points) What code you need to write in `llist.h` to protect it against double-inclusion?

Solution:

```
#ifndef LLIST_H
#define LLIST_H

/* some content ... *

#endif
```

b) (6 points) Write a makefile to compile the program into the executable named `llist` when we run the command '`make`' or '`make all`'. The make should do separate compilation of files `main.c` and `llist.c`.

Solution:

```
# makefile for the program llist
.PHONY: all
all: llist                              2pt (must be first target)

llist: llist.o main.o                   2pt
        gcc -o llist llist.o main.o

llist.o: llist.c llist.h                1pt
        gcc -c llist.c

main.o: main.c llist.h                  1pt
        gcc -c main.c

clean:
        rm rpn rpn.o stack.o
```

Note: Including the special '.PHONY' target, and the 'clean' target are not mandatory for full marks.

10. (10 points) Give brief answer.

a) (5 points) Briefly describe what the `malloc` function does and its time efficiency.

Solution: The malloc function reserves a block of memory on the heap, of the required size and returns a pointer to it. (Optional: The function also records the size of the block in a memory location just before the block.) The function searches for a block of the appropriate size by traversing a linked list of free blocks. If it does not find a block of the appropriate size, it uses a system call to request more memory from the operating system kernel, and adds this memory to the heap. If it does not succeed to get appropriate additional memory, it returns NULL. The malloc function is typically fast, but it may take some time if the list of the free blocks is very long, and if it needs to execute the system call.

Marking scheme:
1pt - reserves a block of memory
1pt - size given as parameter
1pt - returns a pointer
1pt - returns NULL if not successful
1pt - traverses a linked list of blocks, so it may take some time

- may need to call system call for more memory (if this is mentioned it is a bonus 1pt, but total cannot go over 5pt)

b) (5 points) Briefly describe the `mergesort` algorithm, its advantages and disadvantages, and discuss running time. If you want to use code, use pseudo code.

Solution: The mergesort algorithm can be applied to sort arrays, but also linked lists. The algorithm works by dividing the array into two halfs, recursivelly sorting each half by invoking itself, and then merging the sorted half into the final array. The base recursion case is when the array has only one element. The merge operation works by reading through two sorted arrays from the start in parallel, comparing the two next elements in the arrays, and copying and proceeding with the element that comes first in the sorting order. The algorithm is an efficient algorithm since it has $O(n \log(n))$ running time. An advantage is that it works very well with linked lists (and large files). It is also a stable sort algorithm. A disadvantage is that it may not be as efficient in practice on arrays as the quicksort.

Marking scheme:
2pt - for showing understanding of merge sort (2 operations: recursive split, and merge
1pt for a valid advantage
1pt for a valid disadvantage
1pt for correct running time

11. (10 points) Code snippets

(10 points) Write a C program that reads a sequence of integers, one integer per line, from the standard input. For each input integer n, such that $n \geq 1$, the program must print the sum of all positive integers k such that $1 \leq k \leq n$. The input will contain one integer less than 1, and the program must terminate once it reads this integer.

Solution:

```
#include <stdio.h>

int main() {
  int n, k, sum;

  while (1==scanf("%d", &n) && n>=1) {
    sum = 0;
    for (k=1; k<=n; k++)
      sum += k;
    printf("%d\n", sum);
  }
  return 0;
}
```

12. (9 points) Shell scripting

a) (3 points) Briefly explain how shell interprets double parentheses in a command, such as: `((` *some expression* `))`

Solution: The double parantheses in bash shell are interpreted as arithmetic expressions, or as an arithmetic for-loop if preceeded with the command 'for'.

b) (6 points) Briefly explain the lines (1), (2) and (3) in the shell script below:

```
if [ ! -d $1 ]; then
   echo Error                              (1)
   exit 1
fi

if [ -f $1/testfile.txt ]; then            (2)
  if [ ! -e tmp/testfile.txt ]; then       (3)
      cp $1/testfile.txt tmp/testfile.txt
  fi
fi
```

Solution: The line (1) is an 'echo' command used to echo an 'Error' message as a result of a previous test. Additional explanation: If the first argument given to the script (`$1`) is not a directory, the error will be printed and the script will exit.

The line (2) tests whether there exists a regular file with the name `$1/testfile.txt`.

The line (3) tests whether there does not exist a file with the name `tmp/testfile.txt`.

The lines (2) and (3) ensure that it is possible to copy the file `$1/testfile.txt` to the file `tmp/testfile.txt`, without overwriting an existing file.

13. (8 points) File operations in C

(8 points) Consider the following two choices for opening a file:

```
FILE *fp = fopen("foo", "ab+");  /* choice A */
FILE *fp = fopen("foo", "a+");   /* choice B */
```

and the following two choices for writing the `double` variable `x` to the file:

```
fprintf(fp, "%lf\n", x);            /* choice C */
fwrite(&x, sizeof(double), 1, fp);  /* choice D */
```

a) Which choices for opening the file and writing to the file are appropriate to be used together? Why?
b) Give one advantage for each choice of writing `x` to the file.

Solution: a) The appropriate choices are: A-D and B-C, because in A-D we assume that the file is binary, and in B-C we assume that the file is a textual file. [4pt = 2pt for choices, 2pt for explanation]

b) An advantage of binary file is that we save on disk space (memory) because the representation is more compact. (Another advantage is that we do not loose on precision.) [2pt]

An advantage of textual file is that the representation is more clear and easy to inspect externally. (Another advantage is that the representation is more portable because it does not depend on an internal memory representation of floating-point numbers.) [2pt]

14. (32 points) Write a C program.

(32 points) Assume that you are working on a program to process product prices. The data will be stored in a linked list where the structure of a node is given in C as follows:

```
struct node {
  char prod[50];      /* prod  is product name */
  double price;       /* price is product price */
  struct node *next;  /* next  is pointer to next node */
};
```

(a) (4 marks) Write a C function `printprod` that takes a pointer to the above node structure and prints a line with the following: the product name, a comma (`,`), the product price, another comma, and a newline character to the standard output. An output example is: `office desk, 356.78,`
The function prototype must be: `void printprod(struct node *n);`

Solution:

```
void printprod(struct node *n) {
  printf("%s, %lf,\n", n->prod, n->price);
}
```

(b) (10 marks) Write the C function `readprod` which allocates a node structure, and fills it with a product name, price, and sets member `next` to NULL. The product name and price are read from the standard input. The input format is as follows:
`orange juice, 5.95, ....`
so, the product name consists of some characters that are not comma, followed by a comma, followed by product price, followed by comma, and some other optional characters in a line. You can assume that product name is not longer than 49, and the function should also read all characters to the end of line but not store them anywhere. You do not need to do much error checking except that if you cannot read input in required format the function should return NULL. The function returns a pointer to the structure created like this. The function prototype is: `struct node* readprod();`

Solution:

```
struct node* readprod() {
  struct node *r = malloc( sizeof(struct node) );              /* 2pt */
  if (2 != scanf("%[^,],%lf", &(r->prod[0]), &(r->price))) {  /* 4pt */
    free(r); return NULL;                                      /* 1pt */
  }
  r->next = NULL;                                              /* 1pt */
  scanf("%*[^\n]\n");                                          /* 2pt */
  return r;
}
```

(c) (10 marks) Write the C function `insert` which inserts a node into a linked list. You can assume that list has products sorted by price from high to low, and after inserting the node, the list should remain sorted. The first argument `head` is the pointer to the first element of the list, or NULL if the list is empty. The second argument `newnode` is the new node to be inserted. The function returns the new head of the list. The function prototype is: `struct node* insert(struct node *head, struct node *newnode);`

Solution:

```
struct node* insert(struct node *head, struct node *newnode) {
  struct node *prev, *p;
  if (head==NULL) return newnode;                    /* 1pt */
  if (newnode->price > head->price) {                /* 2pt */
    newnode->next = head; return newnode;
  }
  prev = head; p = head->next;                       /* 1pt */
  while (p != NULL && p->price > newnode->price) {   /* 3pt */
    prev = prev->next; p = p->next;
  }
  prev->next = newnode; newnode->next = p;           /* 2pt */
  return head;                                       /* 1pt */
}
```

(d) (8 marks) Write a C program that reads a sequence of products and prints them out sorted by their price from high to low. The program must read the the products using the function `readprod` defined in (b), insert them into a linked list using the function `insert` defined in (c). The products must be printed using the function `printprod` defined in (a). Write a complete program with includes and the function main, but you can assume that the functions `printprod`, `readprod`, and `insert` are already defined and their prototypes included, and `struct node` already defined.

Solution:

```
#include <stdio.h>                                  /* 1pt */

int main() {
  struct node *head=NULL, *h;                       /* 1pt */
  while (NULL != (h = readprod()))                  /* 2pt */
    head = insert(head, h);                         /* 1pt */
  for (h = head; h!=NULL; h = h->next)              /* 2pt */
    printprod(h);                                   /* 1pt */
  return 0;
}
```

A tentative marking scheme is included on the side. An include such as stdio or stdlib is required for NULL to be defined.