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) Multics is an advanced version of Unix, developed from Unix version 4.

Solution: **False.** (Multics was a predecessor of Unix.)

b) (2 points) The `cut` command is used to display first `n` lines from a text file.

Solution: **False.** The `cut` command is used to print some columns of data from a text file.

c) (2 points) In a Unix-style system, each process is assigned a memory space, divided into the following four parts: code, stack, heap, and queue.

Solution: **False.** Instead of 'queue' it should be 'data'.

d) (2 points) The `calloc` C function is used to make a system call to the kernel of the operating system.

Solution: **False.** The `calloc` function is used in memory allocation.

e) (2 points) The `gdb` command `break` is used to mark a line in the program where program execution will stop and give us a chance to examine the state of the variables.

Solution: **True.**

f) (2 points) The following C code "`char *f="%d\n"; printf(f, 10);`" is valid and prints the number `10`, followed by a newline character.

Solution: **True.**

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

a) (3 points) If `s1` and `s2` are two string variables, and `i` is an integer variable; which of the following lines is NOT a valid use of string functions?

A. `strcmp(i, s1);`

B. `strcpy(s2, s1);`

C. `strcat(s1, s2+i);`

D. `i = strlen(s1);`

Solution: **A.** We cannot compare a string with an integer.

---

b) (3 points) Unix has a quite general concept of a file, including several different file types. Which of the following is NOT a file type in Unix?

A. symbolic link

B. job

C. socket

D. directory

Solution: **B.** A job is a shell concept, not a file type. The others are file types.

---

c) (3 points) If `p` and `q` are pointers to `int`, pointing to elements of an array, which of the following statements is not valid:

A. `p = q + 2;`

B. `*p = p - q;`

C. `*p = p + q;`

D. `*p = *p + *q;`

Solution: **C.** We cannot add two pointers.

---

d) (3 points) Which of the following lines is a correct way to use the function `scanf`?

A. `int i=0; scanf("%d", i);`

B. `int i=0; int *p=&i; scanf("%d", *p);`

C. `char s[10]="1234567"; scanf("%3s", s);`

D. `char c; scanf("%c", c);`

Solution: **C.** `scanf` expects address (i.e., pointer) as an argument.

3. (10 points) Program Output (pointers).

What is the output of the following program? Include notes or diagrams that justify your answer.

```
int a[10] = {1, 20, 3}, i;
int *p = &a[3];
int *q = p+2;

for (i=3; i<10; i++) a[i] = a[i-2]+1;

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

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

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

Solution: Short answer:

```
 -15 3 16
 4 22 4 4
```

[Marking scheme: 6 points first row, 4 points second row]

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

```
   0  1 2 3 4 5 6 7 8 9
a: 1 20 3 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 20 3 21 4 22 5 23 6 24
           p    q
```

After the next line with pointer arithmetic:

```
   0  1 2  3 4  5 6  7 8  9
a: 1 20 3 21 4 22 4  4 6 24
                p      q
```

Now, the first printf command will produce:

```
-15 3 16
```

because: $1 + 6 - 22 = -15$, $8 - 5 = 3$, and $22 - 6 = 16$.
The next for-loop prints the value of array `a` from 4 to 7, which are:

```
(i=4)4 (i=5)22 (i=6)4 (i=7)4
```

4. (7 points) Program Output (strings). What is the output of the following program? Include notes or diagrams that justify your answer.

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

int main() {
  char s[90];
  char *p = s;
  strcpy(s, "12345-");
  strcat(p, "ABCDE");
  p += 6;
  printf("%d %d %d\n", strlen(s), strlen(p), p-s);
  *(--p) = '\0'; p+=1;
  printf("(%s) (%s)\n", s, p);
  strcat(s,"XYZ");
  printf("(%s) (%s) (%s)\n", s, p+3);
  return 0;
}
```

Note: The line `printf("(%s) (%s) (%s)\n", s, p+3);` was supposed to be `printf("(%s) (%s) (%s)\n", s, p, p+3);`. The students were instructed to cross out one `(%s)`, or any other reasonable assumption based on demonstrated knowledge of `printf`, strings, and pointers would be acceptable.

Solution: As stated:

```
11 5 6                          3pt
(12345) (ABCDE)                 2pt
(12345XYZ) (DE)                 2pt
```

and printf would make a Segmentation Fault, but mentioning Segmentation Fault is not required. If one `%s` is crossed-out, there would be not Segmentation fault. The original intention was to include `p` (`s, p, p+3`) which would produce:

```
11 5 6                          3pt
(12345) (ABCDE)                 2pt
(12345XYZ) (YZ) (DE)            2pt
```

Explanation: After the first five lines of the main function (before `printf` call), the content of the array `s` and pointer values are:

```
s: 1 2 3 4 5 - A B C D E \0
               p
```

So after the printf statement, the output is:
`11 5 6`
because the lenght of 's' is 11, the lenght of 'p' is 5 (to the null character), and the distance between p and s is 6. After the next line with moving p back, setting *p to the null character, and moving p forward again, we have:

```
s: 1 2 3 4 5 \0 A B C D E \0
                 p
```

so the strings s and p produce output:
`(12345) (ABCDE)` after the next line we concatenate `"XYZ"` to 's' and get:

```
s: 1 2 3 4 5 X Y Z \0 D E \0
               p
```

so printing strings 's' and `p+3` (which starts at letter 'D') gives:
`(12345XYZ) (DE)`

Printing strings 's', 'p', and `p+3` would give:
`(12345XYZ) (YZ) (DE)`

5. (4 points) Program Output (strcmp). What is the output of the following program? Include notes or diagrams that justify your answer.

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

void pr(char *s, char *p) {
  int i = strcmp(s,p);
  if ( i == 0 )
      puts("strcmp == 0");
  else if ( i  < 0 )
      puts("strcmp < 0");
  else
      puts("strcmp > 0");
}

int main() {
  char s[9] = "abc-78";
  char *p = "abf-192";

  pr(s, p); pr( s + 3, p + 5);
  return 0;
}
```

Note: `pr( s + 3, p + 4);` was supposed to be `pr( s + 4, p + 5);`. The students were instructed to make this change, but using '3' is also fine, the problem is just a bit harder since we need to remember that '`-`' comes before digits in the ASCII order. Even if student does not remember the order but notices that it is relevant makes the answer correct.

Solution:

```
strcmp < 0
strcmp < 0
```

This is solution for both cases `s+3` and `s+4`.
Marking scheme: 2 pt each.

If student mentiones that they are not sure about the ASCII code of `-` they should get 2pt for the second part. However if both are `> 0`, 1 pt since it may be a simple confusion about sign of strcmp

6. (10 points) Single command line. For each of the following questions, write a single Unix command line to perform the task required. You can use pipes and the list of allowed commands are: cut, ls, grep, egrep, sort, uniq, wc

a) (3 points) Print a list of files in the directory `../dir1` that have names starting with an uppercase letter and ending with '`.txt`'.

Solution:

```
ls ../dir1/[A-Z]*.txt
```

b) (3 points) The file `words.txt` contains one word in each line. Print the number of words in this file that are six or seven characters long, start with 'p' or 'b' and end with 'ing'.

Solution:

```
grep '^[pb]...?ing$' words.txt | wc -l
```

b) (4 points) The file `games.txt` contains a list of hockey games in the following format "*team1*`:`*team2*`=`*score1*`:`*score2*" in each line (example: `Jets:Sabres=3:2`). If the first team in each game is the home team, and the second team is the visiting team, write a command to print out all visiting teams, sorted without repetition.

Solution:

```
cut -d '=' -f 1 games.txt | cut -d: -f 2 | sort | uniq
```

7. (8 points) Give concise answers (permissions, code).

a) (4 points) What permissions are assigned to the file '`f.sh`' with the command: `chmod 673 f.sh` ? Can the user (owner) execute the file?

> **Solution:** `rw- rwx -wx` [3pt]
> No, user cannot execute the file. [1pt]

b) (4 points) In the following function, `N` is a compile-time constant:

```
int f(int a[][N]) {
  printf("%d\n", &a[0][0] - &a[2][6]);
}
```

If the function prints the number -20, what is the value of `N`? Explain.

> **Solution:** `N = 7` Explanation, `a[0][0]` is the first element of the multidimensional array, and `a[2][6]` is the seventh element of the third row, so it is `2*N+6` further in the memory. Since this is 20 positions in the memory, then N must be 7.

8. (9 points) Give concise answers (testing, git/svn).

a) (4 points) What is the difference between white and black box testing?

Solution: In white box testing, we are using our internal knowledge of an implementation to achieve maximum code coverage by test cases.

In black box testing, we use requirements (specification) to choose test cases to achieve the best coverage of the specification.

b) (5 points) When we submit changes in 'git' to a remote repository, we use one additional command compared to when doing the same operation in 'svn'. This is related to an essential difference between 'git' and 'svn'. Briefly explain this difference.

Solution: 'git' is a distributed source version control system and 'svn' is centralized. [2pt]

In 'git' we first commit the changes to the local repository, and then we 'push' them to the remote repository. In 'svn' we immediatelly commit to the remote, central repository. [3pt]

9. (8 points) Large program organization.

(8 points) A C program consists of three files `main.c`, `aux.c`, and `btree.c`, and a common header file `b.h`, which is used in all C files. The program executable is called `btree`. Write Makefile rules for the targets: `btree`, `main.o`, `aux.o`, and `btree.o`, for modular compilation and linking of the program.

Solution:

```
btree: main.o aux.o btree.o                          # 3.5pt
        gcc -o prog main.o aux.o. btree.o
main.o: main.c b.h                                   # 1.5pt
        gcc -c main.c
aux.o: aux.c b.h                                     # 1.5pt
        gcc -c aux.c
btree.o: btree.c b.h                                 # 1.5pt
        gcc -c btree.c
```

(Round final points if .5 to up.)

10. (11 points) Give brief answer.

a) (6 points) Briefly describe the problem of fragmentation of heap. Make sure to describe two types of fragmentation mentioned in class.

Solution: Due to arbitrary order of allocation and deallocation of memory block and their varying size, the heap memory (free store) may become fragmented; i.e., it may consists of many small "holes" of free memory. Because of this it may take a long time to allocate a new memory block. It may also be impossible to find a block of appropriate size, even though there may be sufficient free memory all-together.[4pt]

There are two types of fragmentation: *external fragmentation* and *internal fragmentation.*[1pt]

[1pt to describe each]

The external fragmentation is what was just described: the free memory being fragmented into many small blocks so it is not possible to find larger continuous block of free memory.

The internal fragmentation happens since the blocks are allocated in certain multiples of bytes; e.g., 8 bytes. Because of this, any request of memory is rounded up to this multiple, and more memory is allocated than actually used.

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

Solution: The function realloc takes two parameters: a pointer to an already allocated block of the heap memory, and an integer that is the new size of the block. The function resizes the block to the newsize and returns a pointer to the new block. (2pt)

The contents of the old block are preserved as much as they fit in the new block. In this way, we can enlarge the block or shrink it. (1pt)

If the new size is 0, the function is equivalent to the free function, and if the original pointer is NULL, the function is equivalent to the function malloc. (1pt)

The worst-case running time of the function is linear in the block size, since it may need to copy the block, in addition to time needed to find the free block on the heap, similarly to the malloc function. However, realloc is generally more efficient to use than using a combination of the function malloc, copying, and the function free, since realloc may be able to resize the block without the need for copying the contents. (1pt)

11. (8 points) Brief Answers and Code snippets

a) (4 points) Briefly describe the implicit type conversion in an assignment. Give an example.

Solution: The implicit type conversion in an assignment means that the value of the right expression in an assignment is coverted to the type of the expression on the left of the assignment. For example, if `i` is an integer variable, then `i = 4.3;` will result in 4.3 being converted to 4 and 4 is assigned to `i`.

b) (4 points) The following function reads some characters from the standard input and stores them in a string on a heap without wasting space. Fill in the missing code and explain how much of input it will store.

```
char *f() {
  int ch, size = 100, count=0;
  char *b = malloc( size );
  while (EOF != (ch = getchar())) {

    if (count > size - 2)

      b = ______________ (b, size *= 2);  /* Fill in blanks * /

    b[count++] = ch;
  }

  b[count] = '\0';

  b = _________________ ( b, ______________________ );
                                          /* Fill in blanks */
  return b;
}
//How much input is read?
```

Solution: Parts to be filled:

```
      b = realloc  (b, size *= 2);     // 1pt
...

  b = realloc  (b,   strlen(b) + 1  );  // 1pt + 1pt
```

It will store all input. [1pt]

Comments: The last question was ambiguous. The intention was to test whether studens recognize that the function reads whole standard input, but due to ambiguity the following answers are also considered as correct: `count` characters, or until `EOF` is read.

In the fill-in part (3) another accepted answer is `count+1`, however in both cases one extra character for the null character is critical — that point is not given unless this is taken into account.

12. (10 points) Shell scripting

Given the bash script below, briefly explain the lines (1) to (7). Explain the purpose of this script based on your observation. [2pt]

```
if (( $# != 2 )); then                          #(1) 2pt
   echo Error
   exit 1
fi

for (( i = $1; $i <= $2; i = $i + 1 )) do       #(2) 2pt
  if [ -f test$i.in ]; then                     #(3) 1pt
    ./a.out < test$i.in > test$i.new            #(4) 1pt
    diff test$i.out test$i.new                  #(5) 1pt
  fi                                            #(6) 0.5pt
done                                            #(7) 0.5pt
```

Solution: The line (1) checks whether the script received two command line arguments. Additional explanation: If the script did not receive exactly two command-line arguments, we print Error and exit the script.

The line (2) is a for-loop that expects two command-line arguments to be two numbers, and sets variable `i` for all numbers in the range of these two numbers.

The line (3) checks whether the file `test$i.in` exists, where `$i` is number in the specified range. In at least one of the lines (3–5) it should be mentioned that `$i` is replaced with a number in the range.

Line (4): If the file `test$i.in` exists, where `$i` is a number, then we run the program `./a.out` with input `test$i.in` and produce output in `test$i.new`.

Line(5): We compare output in `test$i.new` with the file `test$i.out`

Line (6): End of if-statement.

Line (7): End of for-loop.

Purpose of the script: The script can be used to test a program compiled in `./a.out` with prepared test cases in files named such as `test1.in` for prepared input and `test1.out` for prepared output, and so on (such as `test2.in`, `test2.out`).

13. (9 points) File operations in C

(9 points) Briefly explain the standard function `fopen`. What are the parameters and what the function returns?

Solution: The function `fopen` is used to open files. [1pt]
The first argument is the file name, [1pt]
and the second argument is the file mode. [1pt]
The file mode can have letters 'r', 'w', or 'a' for read, write, or append. [3pt]
It can contain character '+' for read and write. [1pt]
It is assumed that the file is textual, unless it contains the letter 'b' which denotes the binary file. [1pt]

The function returns a pointer to a structure FILE, which contains data about open file. [1pt]
NULL is returned if opening the file was not successful.

14. (13 points) Write a C program.

Assume that you are working on a program to process games of hockey teams and collect their points. The data is organized in a linked list where a node structure is:

```
struct node {
  char *team;         /* team    is the team name */
  int  points;        /* points  are accumulated team points */
  struct node *next;  /* next    is pointer to next node */
};
```

(a) (4 points) Write a C function `teamPrint` that takes a pointer to the above node structure and prints a line with the team name, the number of points, and a new line. For multiple lines to be aligned, print team name to at least 12 characters width, and points to at least 2 digits width, with a space between them. An output example is " `Maple_Leafs  4`". The start of the function is:
`void teamPrint(struct node *n) {`

Solution:

```
void teamPrint(struct node *n) {
  printf("%12s %2d\n", n->team, n->points);
}
```

(b) (9 points) Write the C function `teamNew` which allocates a node structure, sets the team name `team` to given name `tname`, sets `points` to 0, and the `next` pointer to NULL. The team name should also be allocated on heap and copied from the given name since the given name may have temporary life span. No error checking is required. The function returns the pointer to the new node. The start of the function is:
`struct node *teamNew(char *tname) {`

Solution:

```
struct node *teamNew(char *tname) {
  struct node *r = malloc( sizeof(struct node) );  /* 2pt */
  r->team = malloc( strlen(tname) + 1 );           /* 2pt */
  strcpy(r->team, tname);                          /* 2pt */
  r->points = 0;                                   /* 1pt */
  r->next = NULL;                                  /* 1pt */
  return r;                                        /* 1pt */
}
```

15. (10 points) Write a C program (continued)

(continuation of the previous question) Write a C function `frontPull` which takes a head pointer of a linked list `head` and a team name `tname` as the arguments. If the linked list contains a node with the team name `tname`, the function moves the corresponding node to the start of the list (head of the list). If the list does not contain a node with such name, a new node is created using the function `teamNew` from the previous page, and the new node is added to the start of the list. The head pointer of the new list is returned. The function prototype is: `struct node* frontPull(struct node* head, char* tname);`

Solution:

```
struct node* frontPull(struct node* head, char* tname) {
  struct node *prev, *p;
  for (prev = NULL, p = head; p != NULL; prev = p, p = p->next) {/*3pt*/
    if (strcmp( p->team, tname ) == 0)
      break;
  }

  if (p == NULL) {                                        /* 3pt */
    p = teamNew(tname);
    p->next = head;
    return p;
  }
  if (prev == NULL)                                      /* 1pt */
    return head;

  prev->next = p->next;                                  /* 3pt *
  p->next = head;
  return p;
}
```

16. (10 points) Write a C program (continued)

(continuation of the previous question) Write a C function `frontPush` which takes a head pointer of a linked list `head` as an argument. Assume that the linked list is sorted **starting from the second element of the list** by `points` in decreasing order (from high to low). The function moves the first element to an appropriate position in the list so that the whole list is sorted by points in the decreasing order. The head pointer of the resulting list is returned. The function prototype is: `struct node* frontPush(struct node* head);`

Solution:

```
struct node* frontPush(struct node* head) {
  if (head==NULL || head->next == NULL ||                    /* 3pt */
      head->points >= head->next->points )
    return head;
  struct node *prev=head, *p=head->next, *oldhead=head;
  head = head->next;
  while (p != NULL && p->points > oldhead->points) {        /* 3pt */
    prev = p; p = p->next;
  }
  oldhead->next = p;                                        /* 3pt */
  prev->next = oldhead;                                     /* 1pt */
  return head;
}
```

17. (12 points) Write a C program (continued)

(continuation of the prev. question) Write a C program (just the function `main`) that uses the previous functions to read game scores and print team names with their total points sorted in a decreasing order of points. You can assume that team names have no spaces and are not longer than 99 characters, there are no tie games, and a team gets 2 points for a win and 0 points for a loss. The input format is shown below. Decide when input ends by using the return value of the `scanf` function. The output should use the function `teamPrint` to print output as shown below. If some teams have the same number of points, their relative order is not important (it is random).

Sample Input	*Sample Output*
`Bruins : Maple_Leafs = 2:4`	`Maple_Leafs 4`
`Bruins : Panthers = 0:5`	`Penguins 2`
`Jets : Penguins = 3:6`	`Panthers 2`
`Maple_Leafs : Sabres = 4:3`	`Sabres 0`
	`Jets 0`
	`Bruins 0`

Solution:

```
int main() {
  struct node *head=NULL, *p;
  char team1[100], team2[100];
  int pt1, pt2;                                    /* 2pt */
  while (4==scanf(" %99s : %99s = %d : %d", team1,team2, &pt1,&pt2)) {
    head = frontPull(head, team1);                 /* 2pt */
    if (pt1 > pt2)
      head->points += 2;                           /* 1pt */
    head = frontPush(head);                        /* 1pt */
    head = frontPull(head, team2);                 /* 2pt */
    if (pt1 < pt2)
      head->points += 2;                           /* 1pt */
    head = frontPush(head);                        /* 1pt */
  }

  for (p=head; p!=NULL; p = p->next)               /* 2pt */
    teamPrint(p);

  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.