University of Toronto Scarborough

CSCB09 Summer 2019 Midterm Test

Duration - 1 hour 15 minutes

Last Name: Student Number:

First Name: UTORid:

1:	/14
2:	/16
3:	/10
Total:	/40

There are technical reminders after the questions.
There are blank pages for scratch work at the end.

Blank page for sketch work.

1. The prof wishes to identify 3 students who have the 3 highest marks so as to reward them! The prof has the following input files:

   - Student file: Each line has login name, last name, first name, and email address, separated by single commas. The lines are sorted alphabetically by login names. There are no duplicates. Sample line:

     ```
     norrisc,Norris,Carrie,cnorris@utoronto.ca
     ```

   - Mark file: Each line has login name and mark, separated by a single colon. The lines are sorted alphabetically by login names. There are no duplicates. Every student in this file is present in the student file. (The converse may be false.) Marks are not blank. Sample line:

     ```
     norrisc:98
     ```

   (a) [7 marks] Use a single Bourne shell pipeline to compute and output the 3 students who have the 3 highest marks; use only the utilities listed on page 8. The lines should be sorted by marks in decreasing order. The filename of the student file is 'students.csv'. The filename of the mark file is in the variable `$mark`. Assume that different students get different marks. The output should go to stdout. The output format should go like this:

   ```
   norrisc,Norris,Carrie,cnorris@utoronto.ca,98
   pittj,Pitt,John,jpitt@gmail.com,96
   fordk,Ford,Kristen,kford@hotmail.com,95
   ```

mark=file.txt
sort -t: -k 2 -n $mark | head -n 2 | cat j.csv

tr : , < markfile | sort -t, | join -t, -o 2.1,2.2,2.3,2.4,1.2 - studentfile | sort -t, -rk 5 -

(b) [7 marks] The prof actually has multiple mark files and wishes to run the above on each mark file. The prof wishes to run a Bourne shell script and give the filenames of the mark files as command line arguments, e.g.,

```
sh best3.sh b09a1marks d65test2marks d85a2marks
```

The shell script should check that a mark file is a regular file and is readable, before running the pipeline; if not, output the message "cannot be read". But no need to check 'students.csv'. All outputs should go to stdout. The output format should go like this:

```
b09a1marks:
norrisc,Norris,Carrie,cnorris@utoronto.ca,98
pittj,Pitt,John,jpitt@gmail.com,96
fordk,Ford,Kristen,kford@hotmail.com,95
d65test2marks:
cannot be read
d85a2marks:
cartera,Athena,Carter,acarter@yahoo.com,84
leenaomi,Lee,Naomi,nlee@utoronto.ca,77
tramph,Tramp,Hugo,htramp@utoronto.ca,72
```

You may write "<part (a)>" where you would put the pipeline from part (a).

2. (a) [6 marks] Write a C function to evaluate a given polynomial at a given point.

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

The coefficients a_i are given in `a[i]`. Assume $n \geq 0$.

Note:

$$a_0 + a_1 x + a_2 x^2 + a_3 x^3 = (((a_3)x + a_2)x + a_1)x + a_0$$

so you don't need exponentiation, you just need multiply-and-add. This should take linear time and absolutely no need to compute or cache x^i.

```
double eval(double x, int n, double *a)
{
```

```
int horner(int poly[], int n, int x)
{
    int result = poly[0];  // Initialize result

    // Evaluate value of polynomial using Horner's method
    for (int i=1; i<n; i++)
        result = result*x + poly[i];

    return result;
}
```

(b) [2 marks] It is more convenient and organized to define a struct to keep together the degree and the coefficients of a polynomial:

```
typedef struct poly
{
  int n;       /* degree */
  double *a;   /* coefficients */
} poly;
```

Write a C function to evaluate a given polynomial at a given point, but this time the polynomial is given by a pointer to the struct `poly`. You are encouraged to call `eval`.

```
double eval_poly(double x, struct poly *p)
{
```

```
    int result = p->a[0];  // Initialize result

    // Evaluate value of polynomial using Horner's method
    for (int i=1; i<p->n; i++)
        result = result*x + p->a[i];

    return result;
}
```

(c) [8 marks] The following code fragment reads a polynomial (reads the degree, then reads the coefficients in the order from a_0 to a_{n-1}), reads a real number x, then evaluates the polynomial at x. It is just missing space allocation, how/where to store coefficients, and space deallocation. Fill in the missing parts.

```
int n, i;
double x;
struct poly *p

scanf("%d", &n);
/* Fill in code below to malloc space for poly and
 * coefficients. Set p to the poly space.
 */
```

p= poly * malloc(sizeof(poly)*n);

```
for (i = 0; i < n; i++) {

  /* Fill in the parameter below. */

  scanf("%lf",
```

&(p[i])

```
);

}
scanf("%lf", &x);
printf("%f\n", eval_poly(x, p));

/* Fill in code below to free the space you malloc'ed. */
```

free(p);

3. A circular linked list uses the same node definition as singly linked lists:

```
typedef struct node
{
  int i;
  struct node *next;
} node;
```

In a singly linked list, the last node's 'next' pointer is NULL. In a circular linked list, the last node's 'next' pointer points back to the first node.

(a) [5 marks] Write a C function to count the number of nodes in a circular linked list.

```
int circular_count(node *head)
{
    Int n=0;
    node * curr=head;
    while(curr!=headIIcurr!=NULL){
    curr=curr->next;
    n++;
    }
    return n;
```

(b) [5 marks] Write a C function to change a singly linked list to a circular linked list.

```
void singly_to_circular(node *head)
{
  node * curr=head;
  while(curr->next!=NULL){
  curr=curr->next;
  }
  curr->next=head;
  //return n;
```

(End of questions.)

Technical Reminders

Utility programs. Convention: *file* can be '-' to mean stdin.

- **cat** [*option*]...[*file*]... : Output concatenation of files and/or stdin. With no files, just stdin.
  -b: number non-empty lines
  -n: number all lines

- **head** [*option*]...[*file*]... : Output the first part of files.
  -n i: the first i lines (default 10)
  -n -i; all but the last i lines

- **join** [*option*]...*file1 file2* : Join lines of two files on a common field.
  -1 n: join on field n (default 1) of *file1*
  -2 n: join on field n (default 1) of *file2*
  -a i: also output unpairable lines from file i
  -e v: replace missing fields with v
  -o *format*: which fields to output; *format* is a comma-separated list of '0' (the join field) or '1.n' (*file1* field n) or '2.n' (*file2* field n); default: join field, then the remaining fields from *file1*, then the remaining fields from *file2*
  -t c: use character c as field separator
  -v i: like **-a** but suppress joined output lines

- **last** [*option*]...[*file*]... : Output the last part of files.
  -n i: the last i lines (default 10)
  -n +i: from line i to last

- **sort** [*option*]...[*file*]... : Sort.
  -k m,n: sort by fields from m to n (as one key)
  -t c: use character c as field separator

- **tee** [*option*]...[*file*]... : Copy stdin to stdout and files.
  -a: append to the files instead of overwriting

- **tr** [*option*]...*set1* [*set2*]: Translate, squeeze, delete.
  E.g., 'tr ab 12' converts 'a' to '1', 'b' to '2'.
  -c: complement *set1*
  -d: delete characters that are in *set1*
  -s: replace consecutive occurrences by single occurrence

- **wc** [*option*]...[*file*]... : Count.
  -c: count bytes
  -l: count lines
  -w: count words

Tests in Bourne shell, e.g., `[ -e path ]`:

- `-e path`: path exists
- `-f path`: path exists and regular file
- `-d path`: path exists and directory
- `-r path`: exists and readable (by you)
- `-w path`: exists and writable
- `-x path`: exists and executable
- `-n string`: string is not empty
- `-z string`: string is empty
- `s1 = s2`: string equality
  Also '`!=`', '`<`', '`>`'
- '`n1 -eq n2`': integer equality
  Also '`-ne`', '`-gt`', '`-ge`', '`-lt`', '`-le`'
- `cond1 -a cond2`: and
- `cond1 -o cond2`: or
- `! cond`: not

C heap memory library:

- `void *malloc(size_t size);`
- `void free(void *ptr);`

Blank page for sketch work. You may tear out for convenience; no need to hand in.

Blank page for sketch work. You may tear out for convenience; no need to hand in.