National University of Singapore
School of Computing

CS2281 **Practical Test 2** Semester 3 04/05

Instructions

PLEASE READ THE INSTRUCTIONS CAREFULLY.

- You have THREE (3) hours to complete this test.
- This is an OPEN book exam, you may refer to your books, notes or any resources online.
- You must answer exam questions independently and may NOT try to obtain outside help of any kind.
- TURN OFF your handphone.
- Answer ALL questions.
- This paper contains FIVE (5) printed pages, including this cover page.

Procedure

- Login to the PC using your NUSNET account as usual.
- You should have received a user id and a password for your special exam account from your invigilator. You must login to sunfire (also known as sf3) using this account.
- **Your code must remain in your special exam account all the time. Transferring of your code to other systems (including laptop, desktop, mail server, web server, file server) is STRICTLY prohibited.**
- You are encouraged to save your work often.
- At the end of this exam, save your files, leave them in your account, and log off. We will collect your files from your exam account after the exam.

Question 1 (50 points)

One of the major source of bugs when writing C program is improper use of string functions. The standard string functions provided by C do not check for buffer overflows, leading to to frequent program crashes and security holes.

In this question, you will implement a C library that provides an easy-to-use string ADT that relief its users from worrying about buffer overflows. To do this, your string ADT must automatically grow and shrink based on the size of the string stored in your ADT.

A string ADT can be implemented using the following

```
typedef struct string {
    char *data;
    int length;
    int capacity;
} string;
```

`data` is an array of characters that stores the string, `length` contains the number of characters in the string, and `capacity` is the size of the `data` array.

The interface to your string ADT is specified below.

string *string_new()

Create and return a new string. Return `NULL` if errors.

void string_cat(string *str1, char *str2)

Concatenate the C string `str2` to your string `str1`. Analogous to `strcat` function in C.

int string_len(string *s)

Return the length of the string `s`.

char * string_to_cptr(string *s)

Return the internal char array that is used to store the string.

void string_printf(string *s, char *fmt, ...)

Analogous to `sprintf` function in C. However, unlike `sprintf` in C, which recognizes format specifier %c, %d, %.3f, etc, `string_printf` only need to recognize two simple format specifier, %s and %d. Furthermore, instead of returning the number of characters printed, as in `sprintf`, `string_printf` does not have to return anything.

A sample program that uses your string ADT is given below:

```
#include "mystring.h"
int main()
{
    int i = 8, j = 10;
    char *x = "goodbye";

    string *s = string_new();

    string_cat(s, "hello ");
    string_cat(s, "world");
    /* next line should print "11 hello world" */
    printf("%d %s\n", string_len(s), string_to_cptr(s));

    string_printf(s, "i=%d j=%d x=%s", i, j, x);
    /* next line should print "i=8 j=10 x=goodbye" */
    printf("%s\n", string_to_cptr(s));

    return 0;
}
```

You should submit four files:

- `mystring.c` which implements your string ADT.
- `mystring.h` which provides type definitions and declarations of functions provided by your ADT.
- `main.c` which contains your test program.
- `Makefile` which compiles `mystring.c` into a static library called `libmystring.a` and links with main.c to produce an executable called `main`.

Question 2 (50 points)

In this question, you will write a shell script called `todo.sh` that implements a todo list. Your todo list should be kept in a text file called TODO in your current directory. Each line in the text file is a todo item, which consists of a deadline and a text description. The deadline is specified by numeric month and day, while the description is a string.

The usage of your script must follows the following description.

- todo.sh
  When your script is called without any arguments, it should print out the current todo list, in increasing order of its deadline. The items are enumerated and each item is numbered 1, 2, 3, etc. according to their order in the output. For example:

```
$ todo.sh
1. 5 21 buy eggs
2. 5 24 do assignment 4
3. 6 1 get birthday present
4. 12 15 buy xmas gift
```

- todo.sh add *month day description*
  Add an item to the todo list with the given month, day, and description.

```
$ todo.sh add 5 29 reply letter from MJ
$ todo.sh
1. 5 21 buy eggs
2. 5 24 do assignment 4
3. 5 29 reply letter from MJ
4. 6 1 get birthday present
5. 12 15 buy xmas gift
```

- todo.sh passed
  Print out all item in the todo list, in increasing order of date, where the deadline has passed (i.e., the deadline is *before* today). The output should be enumerated and preceded by its line number as well.

```
$ todo.sh passed
1. 5 21 buy eggs
2. 5 24 do assignment 4
```

- todo.sh done *line*
  Delete an item from the todo list. Use `line` which is an integer that corresponds to the line number in the printed output of `todo.sh` to specify which item to delete.

```
$ todo.sh
1. 5 21 buy eggs
2. 5 24 do assignment 4
3. 5 29 reply letter from MJ
4. 6 1 get birthday present
5. 12 15 buy xmas gift
$ todo.sh done 2
$ todo.sh
1. 5 21 buy eggs
2. 5 29 reply letter from MJ
3. 6 1 get birthday present
4. 12 15 buy xmas gift
```

You may find the following information useful:

- The command `date +%m` outputs the current month.
- The command `date +%d` outputs today's day.
- `NR` is a predefined variable in `awk` that stores the current line number. Thus,

  ```
  awk '{print NR ". " $0}'
  ```

  reads from stdin, prefixes each input with its line number, and prints to stdout.