COMP9024 25T1 Data Structures and Algorithms

Week 4 Practical
Dynamic Data Structures

1. (Common memory errors)

   Memory errors are common in C programs, and can be difficult to reproduce and debug. You're given a file `memerrors.c` that contains 8 functions, `f1` through `f8`, each of which contains a common memory error.

   You're also given a file `memerrors.txt` that contains a description of 8 possible memory errors:

   ```
   memory leak
   no memory allocated
   buffer over-read
   uninitialised read
   double free
   invalid free
   unchecked malloc
   dangling pointer
   ```

   a. Re-arrange the descriptions in `memerrors.txt` to match the error in each function. For example, the description matching `f1` should appear on the first line, and the description matching `f8` should appear on the last line.

   It is important that you do not modify the text at all, just the order in which each line of text appears. The re-arranged file should only contain 8 lines of text.

   b. Correct each of the 8 functions in `memerrors.c`. Fixing the code will require adding, modifying, moving, or removing one line of code from each of the functions. Once corrected, the code should compile without warnings, run without error, and not leak any memory.

   Here's an example of compiling and running the corrected code:

   ```
   prompt$ gcc -Wall -Werror -std=c11 -O0 -g -o memerrors memerrors.c
   prompt$ ./memerrors
   f1: 10 20 30
   f2: 2024 9024
   f3: 90 24
   f4: 90 24 42
   f5: 90 24
   f6: 2 1 0
   f7: 90 24
   f8: 24 90
   ```

   Here's an example of testing it for memory leaks:

   ```
   prompt$ valgrind -s --leak-check=full --show-leak-kinds=all ./memerrors  > /dev/null
   ==4184052== Memcheck, a memory error detector
   ==4184052== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
   ==4184052== Using Valgrind-3.19.0 and LibVEX; rerun with -h for copyright info
   ==4184052== Command: ./memerrors
   ==4184052==
   ==4184052==
   ==4184052== HEAP SUMMARY:
   ==4184052==     in use at exit: 0 bytes in 0 blocks
   ==4184052==   total heap usage: 9 allocs, 9 frees, 4,160 bytes allocated
   ==4184052==
   ==4184052== All heap blocks were freed -- no leaks are possible
   ```

```
==4184052==
==4184052== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Answer:

a.

```
unchecked malloc
memory leak
buffer over-read
double free
invalid free
uninitialised read
dangling pointer
no memory allocated
```

b.

```
/**
 * Each of the functions f1() to f8() below contains
 * a common memory mistake that results in either
 * a run-time error or "undefined behaviour".
 *
 * First identify each mistake, then correct it.  Fixing the
 * code will require adding, modifying, moving, or removing
 * one line of code from each of the 8 functions.
 *
 * Once corrected, the code should compile without warnings,
 * run without error, and not leak any memory.
 */

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

/* swap the values of two integer variables */
void swap(int *p, int *q) {
   int temp = *p;
   *p = *q;
   *q = temp;
}

/* copy the values of integer array a[] to b[] */
void copy(int a[], int b[], int size) {
   int i;
   for (i = 0; i < size; i++) {
      b[i] = a[i];
   }
}

/* print all the values of a dynamic integer array */
void print(int *b, int size) {
   int i;
   for (i = 0; i < size; i++) {
      printf("%d ", b[i]);
   }
   putchar('\n');
   free(b);
}

/* DO NOT MODIFY ANY OF THE CODE ABOVE THIS LINE */

void f1(void) {
   int ar[3] = {10, 20, 30};
   int *cp = malloc(3 * sizeof(int));
```

```
   assert(cp != NULL);
   copy(ar, cp, 3);
   print(cp, 3);
}

void f2(void) {
   int a = 9024;
   int *c = malloc(sizeof(int));
   assert(c != NULL);
   *c = 2024;
   swap(&a, c);
   printf("%d %d\n", a, *c);
   free(c);
}

void f3(void) {
   int ar[2] = {90, 24};
   int *cp = malloc(2 * sizeof(int));
   assert(cp != NULL);
   copy(ar, cp, 2);
   printf("%d %d\n", cp[0], cp[1]);
   free(cp);
}

void f4(void) {
   int ar[3] = {90, 24, 42};
   int *cp = malloc(3 * sizeof(int));
   assert(cp != NULL);
   copy(ar, cp, 3);
   print(cp, 3);
}

void f5(void) {
   int ar[2] = {90, 24};
   int *cp = malloc(2 * sizeof(int));
   assert(cp != NULL);
   copy(ar, cp, 2);
   print(cp, 2);
}

void f6(void) {
   int ar[3] = {2, 1, 0};
   int *cp = malloc(3 * sizeof(int));
   assert(cp != NULL);
   copy(ar, cp, 3);
   print(cp, 3);
}

void f7(void) {
   int *cp = malloc(sizeof(int));
   assert(cp != NULL);
   *cp = 90;
   int a[1] = {24};
   swap(&a[0], cp);
   printf("%d %d\n", a[0], *cp);
   free(cp);
}

void f8(void) {
   int a = 90;
   int *c = malloc(sizeof(int));
```

```
    assert(c != NULL);
    *c = 24;
    swap(&a, c);
    printf("%d %d\n", a, *c);
    free(c);
}


int main(void) {
    printf("f1: ");
    f1();
    printf("f2: ");
    f2();
    printf("f3: ");
    f3();
    printf("f4: ");
    f4();
    printf("f5: ");
    f5();
    printf("f6: ");
    f6();
    printf("f7: ");
    f7();
    printf("f8: ");
    f8();

    return 0;
}
```

2. (Dynamic linked lists)

   Write a C-program called **llcreate.c** that creates a linked list of integers from user input. Your program should use the following functions:

   - *void freeLL(NodeT *list)*: taken from the lecture.
   - *void showLL(NodeT *list)*: taken from the lecture **but needs modification.**
   - *NodeT *joinLL(NodeT *list, int v)*: returns a pointer to the linked list obtained by appending a new element with data *v* at the end of *list*. **Needs to be implemented.**

   The program:

   - starts with an empty linked list called `all` (say), initialised to `NULL`
   - prompts the user with the message "Enter an integer: "
   - appends at the end of `all` a new linked list node created from user's response
   - asks for more user input and repeats the cycle
   - the cycle is terminated when the user enters any non-numeric input
   - on termination, the program generates the message "Done. List is " followed by the contents of the linked list in the format shown below.

   A sample interaction is:

```
prompt$ ./llcreate
Enter an integer: 12
Enter an integer: 34
Enter an integer: 56
Enter an integer: quit
Done. List is 12-->34-->56
prompt$
```

   Note that any non-numeric data 'finishes' the interaction. If the user provides no data, then no list should be output:

```
prompt$ ./llcreate
Enter an integer: #
Done.
prompt$
```

We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to this exercise. It expects to find a program named* `llcreate.c` *in the current directory. You can use dryrun as follows:*

```
prompt$ 9024 dryrun llcreate
```

Note: Please ensure that your output follows exactly the format shown above.

Answer:

```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

typedef struct node {
   int data;
   struct node *next;
} NodeT;

NodeT *joinLL(NodeT *list, int v) {
   NodeT *new = malloc(sizeof(NodeT));
   assert(new != NULL);
   new->data = v;
   new->next = NULL;

   if (list == NULL) {  // list may be empty
      list = new;
   } else {
      NodeT *p = list;
      while (p->next != NULL) {
         p = p->next;
      }
      p->next = new;
   }
   return list;
}

void showLL(NodeT *list) {
   NodeT *p;
   for (p = list; p != NULL; p = p->next) {
      printf("%d", p->data);
      if (p->next != NULL)
         printf("-->");
   }
   putchar('\n');
}

void freeLL(NodeT *list) {
    NodeT *p = list;
    while (p != NULL) {
        NodeT *temp = p->next;
        free(p);
        p = temp;
    }
}
```

```
int main(void) {
   NodeT *all = NULL;
   int value;

   printf("Enter an integer: ");
   while (scanf("%d", &value) == 1) {
      all = joinLL(all, value);
      printf("Enter an integer: ");
   }
   if (all != NULL) {
      printf("Done. List is ");
      showLL(all);
      freeLL(all);                    // don't forget to free the list
   } else {
      printf("Done.\n");
   }
   return 0;
}
```

Due Date

Tuesday, 18 March, 11:59:59. Late submissions will not be accepted.

Submission

You should submit your files using the following `give` command:

```
prompt$ give cs9024 week4 memerrors.txt memerrors.c llcreate.c
```

Alternatively, you can select the option to "Make Submission" at the top of this page to submit directly through WebCMS3.

Important notes:

- Make sure you spell all filenames correctly.
- You can run `give` multiple times. Only your last submission will be marked.
- Where you're expected to submit multiple files, ensure they form a single submission. If you separate them across multiple submissions, each submission will replace the previous one.
- Whether you submit through the command line or WebCMS3, it is your responsibility to ensure it reports a successful submission. Failure to submit correctly will not be considered as an excuse.
- You cannot obtain marks by e-mailing your code to tutors or lecturers.

Assessment

- Each question is worth 1 mark, for a total of 2 marks.
- Submissions will be auto-marked shortly after the deadline.
- It is important that the output of your program follows exactly the format of the sample output, otherwise auto-marking will result in 0 marks.
- Ensure that your program compiles on a CSE machine with the standard options, i.e. `-Wall -Werror -std=c11`. Programs that do not compile will receive 0 marks.
- Auto-marking will use test cases different to `dryrun`. (Hint: do your own testing in addition to running `dryrun`).

Collection

Once marking is complete you can collect your marked submission using the following command:

```
prompt$ 9024 classrun -collect week4
```

You can also view your marks using the following command:

```
prompt$ 9024 classrun -sturec
```

You can also collect your marked submission directly through WebCMS3 from the "Collect Submission" tab at the top of this page.

Plagiarism

Group submissions will not be allowed. Your programs must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for similar assessments in previous years, if applicable) and serious penalties will be applied, including an entry on UNSW's plagiarism register.

- ***Do not copy ideas or code from others***
- ***Do not use a publicly accessible repository or allow anyone to see your code***

Please refer to the on-line sources to help you understand what plagiarism is and how it is dealt with at UNSW:

- Plagiarism and Academic Integrity
- UNSW Plagiarism Policy Statement
- UNSW Plagiarism Procedure

Reproducing, publishing, posting, distributing or translating this assignment is an infringement of copyright and will be referred to UNSW Conduct and Integrity for action.