Duration: **50 minutes**
Aids Allowed: **1 - 8.5x11 sheet**

**Student Number:** _____

**Last Name:** SOLUTION

**First Name:** _____

**Instructor:** Karen Reid

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

---

This midterm test consists of 4 questions on 8 pages (including this one). *When you receive the signal to start, please make sure that your copy of the test is complete.* Extra space was left for each of the programming questions. Please indicate clearly the part of your work that should be marked.

IMPORTANT: Assume all #includes have been done for you. You do not need to perform error checking unless the question specifically asks you to.

MARKING GUIDE

# 1: _____/ 6

# 2: _____/ 6

# 3: _____/ 9

# 4: _____/ 8

TOTAL: _____/29

*Good Luck!*

## Question 1. [6 MARKS]

Given the following struct definition:

```
struct node {
        int v;
        struct node *next;
};
```

### Part (a) [2 MARKS]

Explain and fix the error in the following code:

```
void f(struct node *n, int v) {
        n->v = v;
        n->next = NULL;
}
int main() {
        int value = 10;
        struct node *newnode;
        f(newnode, value);
        return 0;
}
```

*No memory is allocated for newnode. It can be fixed by adding "newnde = malloc(sizeof(struct node));" after the declaration of newnode in main.*

### Part (b) [2 MARKS]

Explain and fix the error in the following code:

```
void g(struct node *n, int v) {
        n = malloc(sizeof(struct node));
        n->v = v;
        n->next = NULL;
}
int main() {
        int value = 10;
        struct node *newnode;
        g(newnode, value);
        return 0;
}
```

*When g returns, the value of newnode will not have changed. In other words changing the value of n inside g does not propagate out of the function. There are several ways to fix this. One is to return n from g.*

### Part (c) [2 MARKS]

Explain and fix the error in the following code: Bad copy and paste error. The code should have been:

```
struct node *h(int v) {
    struct node n;
    n.v = v;
    n.next = NULL;
    return &n;
}
```

```
void g(struct node *n, int v) {
        n = malloc(sizeof(struct node));
        n->v = v;
        n->next = NULL;
}
int main() {
        int value = 10;
        struct node *newnode;
        newnode = h(value);
        return 0;
}
```

*h returns a pointer to memory that is allocated on the stack. When h returns this memory is no longer reserved for n, so the values at that address will change. A solution is to use malloc inside h to allocate memory for the node on the heap.*

## Question 2. [6 MARKS]

### Part (a) [5 MARKS]

Fill in the type for the following variable declarations so that the variables have the correct type.

| | |
|---|---|
| 1 | `int i;` |
| 2 | `_____ q = &i;` |
| 3 | `_____ j = *q;` |
| 4 | `char a[10];` |
| 5. | `_____ c = a;` |
| 6 | `_____ b = &a[1];` |
| 7 | `_____ d = a[3];` |
| 8 | `char *strs[5];` |
| 9 | `_____ s = strs;` |
| 10 | `_____ t = &strs[3];` |
| 11 | `_____ u = strs[3];` |
| 12 | `_____ v = strs[3][1];` |
| 13 | `_____ w = *strs[3];` |

### Part (b) [1 MARK]

Write down the line numbers that could lead to a memory error (segmentation fault or bus error).

*12, 13*

Solutions:

```
int i;
int *q = &i;
int j = *q;

char a[10];
char *cptr;
char *c = a;
char *b = &a[1];
char d = a[3];
char e = a[3];

char *strs[5];

char **s = strs;
char **t = &strs[3];
char *u = strs[3];
char v = strs[3][1];
char w = *strs[3];
```

## Question 3. [9 MARKS]

Complete the C function below that removes characters in the string `ch` from the string `str`. You may not use any string library functions other than `strlen`

Below is an example of calling `strip`:

```
char input[20] = "Hello there";
strip(input, "eo");
printf("%s", input);  // prints "Hll thr"

void strip(char *str, char *ch) {
```

There are several different ways to solve this problem.

```
void strip(char *src, char *ch) {
    int i = 0;
    int k = 0;
    while(i < strlen(src)) {
        int j = 0;
        while(j < strlen(ch) && ch[j] != src[i]) {
            j++;
        }
        if(j >= strlen(ch)) {
            src[k] = src[i];
            k++;
        }
        i++;
    }
    src[k] = '\0';
}
```

## Question 4. [8 MARKS]

### Part (a) [1 MARK]

Rewrite the following shell loop so that no additional processes are created:

```
for f in `ls /tmp`
```

*for f in /tmp/**

### Part (b) [4 MARKS]

Complete the shell function below that replaces spaces in a string with underscore characters and prints the resulting string. Do not use `tr`, `sed`, or `awk`. Use shell commands we discussed in class such as `set` or `cut`. You will get 3 marks for successfully replacing only the first space. If a string does not have a space the function will print the original string.

For example, calling

```
replace_space "a string with spaces"
```

will produce

```
a_string_with_spaces
```

```
function replace_space {


function rmsp {
    set $1
    result=""
    for str in $*
    do

    if [ -z $result ]
    then
            result=$str
    else
        result=$result"_"$str
        fi
    done
    echo $result

}
```

### Part (c) [3 MARKS]

Using the `replace_space` function, write a program to replace spaces in all the files names in the directory specified by the command line argument.

```
for f in $1/*
do
    rmsp "$f"
done
```

(This page intentionally left blank)

**C function prototypes and structs:**

```
int sprintf(char *s, const char *format, ...)
char *strchr(const char *s, int c)
size_t strlen(const char *s)
char *strncat(char *dest, const char *src, size_t n)
int strncmp(const char *s1, const char *s2, size_t n)
char *strncpy(char *dest, const char *src, size_t n)
char *strrchr(const char *s, int c)
char *strstr(const char *haystack, const char *needle)
```

**Shell comparison operators**

| Shell | Description |
|---|---|
| -d filename | Exists as a directory |
| -f filename | Exists as a regular file. |
| -r filename | Exists as a readable file |
| -w filename | Exists as a writable file. |
| -x filename | Exists as an executable file. |
| -z string | True if empty string |
| str1 = str2 | True if str1 equals str2 |
| str1 != str2 | True if str1 not equal to str2 |
| int1 -eq int2 | True if int1 equals int2 |
| -ne, -gt, -lt, -le | For numbers |
| !=, >, >=, <, <= | For strings |
| -a, -o | And, or. |

Useful shell commands:

du, echo, ls, head, tail, read, sort, uniq, wc
grep (returns 0 if match is found, 1 if no match was found, and 2 if there was an error)
set (Note: with no arguments set prints the list of environment variables) ps aux - prints the list of currently running processes grep -v displays lines that do not match