Written Test - Version A

CSE 2031 3.0

Section E, Fall 2012

Family Name: _____

Given Name(s): _____

Student Number: |___ |___ |___ |___ |___ |___ |___ |___ |___ |

Guidelines and Instructions:

1. This is a 110-minute test. You can use the textbook, but no electronic aids such as calculators, cellphones etc.
2. Answer questions in the space provided. If you need more space, use the back of the page. Clearly indicate that your answer continues on the back of the page.
3. Write legibly. Unreadable answers will not be marked.
4. Keep your eyes on your own work. At the discretion of the invigilators, students may be asked to move.

Question	Out of	Mark
Q1	5	
Q2	10	
Q3	15	
Q4	20	
Q5	20	
Q6	30	
Total	100	
Letter grade		

Q1. [5 marks] The following program compiles and runs with no problems. Indicate what the output of the program is going to be (no explanation necessary).

```
1  #include <stdio.h>
2  main() {
3      int i;  int j;
4      int *p; int *q;
5
6      p = &i;
7      q = &j;
8      i =  1;
9      j =  2;
10     p =  q;
11     *q = 3;
12     printf("%d %d %d %d\n", i, j, *p, *q);
13 }
```

The output will be: 1 3 3 3.

5 marks all or nothing.

Q2. [10 marks] The following program compiles and runs with no problems. Indicate what the output will be if the input is the number 1. **Explain why.**

```
1  #include <stdio.h>
2  #define clone(x) x;x
3  #define remove
4  main()
5  {
6    int i;
7    scanf("%d",&i);
8    clone(i++);
9    switch (i)
10   {
11     case 1: printf("ONE\n");
12     case 2: printf("TWO\n");
13 #ifdef remove
14     case 3: printf("THREE\n");
15 #endif
16     default: printf("OTHER\n");
17   }
18 }
```

The output will be:

```
THREE
OTHER
```

`clone(i++);` will expand to `i++;i++;`. As a result, the value of `i` at the switch statement will be 3.

Since `remove` is defined, the preprocessor directives do not have any effect. As a result, the code after `case 3` will execute. Since there is no `break;`, the code after `default` will execute as well.

No marks if there is no explanation. Partial marks for correct output with incomplete explanation. Even less partial marks for incorrect output, but partially correct explanation.

Q3. [15 marks] The following program attempts to read a string, save it in memory, and then reproduce it on the screen. It compiles with no problems. However, when run it produces a segmentation fault. Your task is to:

(a) Explain the reason for the segmentation fault.

(b) Describe how you would fix the problem.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void allocate(char* p) {
5    p = (char *) malloc (30 * sizeof(char));
6  }
7
8  int main()
9  {
10   int i;
11   char *str;
12   allocate(str);
13   for (i = 0; i < 20; i++) str[i] = getchar();
14   str[20] = '\0';
15   printf("%s", str);
16 }
```

The reason for the segmentation fault is that the `str` pointer does not point to allocated memory, so even after only one character is entered there will be a segmentation fault in line 13.

The `allocate` function appears to perform the necessary memory allocation. However, due to call-by-value, it only has a copy of the `str` pointer, and it cannot change where it points to. As a result, the allocated memory becomes garbage as soon the execution of the `allocate` function finishes.

The problem can be fixed by either inlining the `allocate` function, i.e. copying its code to the main function, or by passing a `char **` to it, such as `&str`.

10 marks for a correct explanation of the problem, 5 marks for the fix. Partial marks are possible if the explanation is incomplete but hints at an understanding of the problem.

Q4. **[20 marks]** The following program attempts to read 40 lines of text, where each line is guaranteed to be exactly 20 characters (including the newline). It compiles with no errors, but produces a segmentation fault when run because it has not allocated any memory.

In your answer, indicate what lines of code you would add and where to fix this problem. For example, say something like: "I would add the line `i++;` between lines 9 and 10.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define LINESNO 40
4  #define CHARNO 20
5
6  int main()
7  {
8    int i,j;
9    char **lines;
10   for (i = 0; i < LINESNO; i++)
11   {
12     char *line = lines[i];
13     for (j = 0; j < CHARNO; j++) line[j] = getchar();
14     line[CHARNO] = '\0';
15   }
16   for (i = 0; i < LINESNO; i++) printf("%s", lines[i]);
17 }
```

The lines string array must be allocated by adding the following line between lines 9 and 10:

```
lines = (char **) malloc (LINESNO * sizeof(char *));
```

Each string must also be allocated. This can be done by adding the following line between lines 11 and 12:

```
lines[i] = (char *) malloc ((CHARNO + 1) * sizeof(char));
```

The `+ 1` in the above line is to ensure enough room for the null character.

10 marks for each added line. Other ways to allocate the memory may be possible and should receive full marks if correct. Partial marks are possible for incorrect solutions that demonstrate an understanding of the issue at hand. No marks for changing the program to use arrays.

Q5. [20 marks] The following program is split across three different files as shown below. Assuming that we attempt to compile it with `cc q5.c q5fun.c`, indicate which one of the following is true:

(a) There is a compile-time error. If so, describe what the problem is (you don't have to describe a fix).

(b) There is a runtime error. If so, describe why this happened (you don't have to describe a fix).

(c) The program compiles and runs with no problems. If so, indicate what the output will be.

q5.c

```
#include <stdio.h>
#include "q5.h"
int number;
main() {
  number = 2;
  printf("%d\n",fun(5));
  printf("%d\n",number);
}
```

q5.h

```
extern int number;
int fun(int);
```

q5fun.c

```
#include "q5.h"
int fun(int x) {
  return x + number++;
}
```

The program compiles and runs with no problems. The output will be:

7

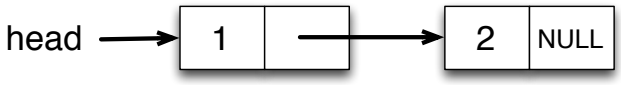
3

20 marks, all or nothing.

Q6. [30 marks] Recall the following linked list data structure that we saw in class.

```
struct list {
  int data;
  struct list *next;
};
```

Assume that a program has used this structure to build the following linked list.

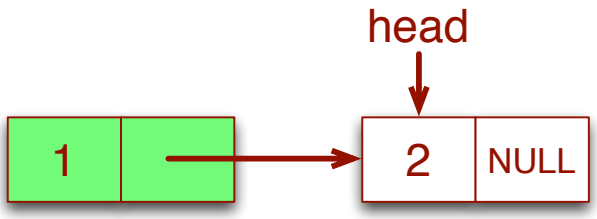


For each of the code segments below and in the following pages, draw a diagram that shows what the list will look like after its execution. If some list element has become garbage, indicate that clearly. The type of variable `head` is `struct list *`.

Segment 1

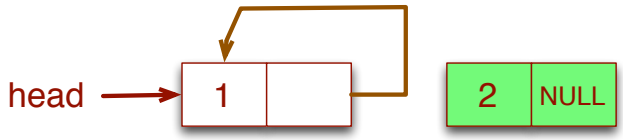
```
head = head->next;
```

6 points, all or nothing, per each segment. Garbage must be shown for full marks. Garbage shown here in green.



Segment 2

```
head->next = head;
```



Segment 3

```
head->next->next = head;
head = head->next;
head->next->next = NULL;
```

Segment 4

```
head->next = NULL;
```

Segment 5

```
head->next->next = head;
```