Student Name:

Student Number:

Faculty of Science
Final Examination

Computer Science 308-206B
Introduction to Software Systems

Examiner: Hans Vangheluwe — Monday, April 30^{th}, 2001

Associate Examiner: Charles Snow — 14:00 – 17:00

INSTRUCTIONS:

1. Answer all questions directly on the examination paper.
2. No notes, books, calculators, computers or other aids of any type are permitted.
3. Translation dictionaries may be used.
4. The exam has 22 questions on 15 pages. The total number of page, including this cover page, is 16.
5. Attempt all questions: partial marks are given for incomplete but correct answers.
6. Numbers between brackets [] denote the weight of each question. The exam is out of a total of 95 points.
7. Use the back of the last page as scrap (it will be ignored during grading). The rear of the other pages may be used as extra space to answer questions.

Good luck !

(1) [2]

What gets printed if the following code fragment is run ?

```
int i=50;
int j=-1;
if ( !(i = j) || (i < 0))
 printf("First option\n");
else
 printf("Second option\n");
```

Why (explain the condition) ?

(2) [3]

Given the declaration

```
char s[] = "Good\tluck again\\%%\n";
```

- What is the length of string `s` (as given by `strlen(s)`) ?
- How many bytes are needed to store the string `s` ?
- What is the return value of `strlen(s)` after the following statement has been executed ?

```
s[(strlen(s)+1)/2] = '\0';
```

(3) [5]

Assume the variable `name` is a character vector. Assume a valid C string is present in `name`. Write a function `count` which returns the number of lower-case letters (a to z) the string contains.

- using iteration:

- using recursion:

(4) [3]

In `mydefs.h`:

```
#ifndef MYDEFS_H_INCLUDED
#define MYDEFS_H_INCLUDED
#define HIDDEN static
#endif
```

In `funcs.h`:

```
#ifndef FUNCS_H_INCLUDED
#define FUNCS_H_INCLUDED
 #include "mydefs.h"
 #define MAX 10
 extern int size;
#endif
```

Write the result of applying the preprocessor (`cpp`) to the file `main.c` which contains

```
#include "mydefs.h"
#include "funcs.h"
int main(void)
{
 HIDDEN int i;
 #undef HIDDEN
 HIDDEN int j;
 return (0);
}
```

(5) [10]

Given the following declarations/initializations

```
int i=10, j=1, k=5;
double x = 30.0, y = 40.0;
char c = 'A';
char message[] = "Hello world\n";
```

Write the *value* and *type* of each of the expressions below. Each expression is independent of the others.

1. `i*(k-j) + sizeof(long int);`

2. `x/i;`

3. `i % k;`

4. `('F' - c)/ 2;`

5. `('e' - 'b')/ 2.0;`

6. `++i*j++;`

7. `message[2] += 2;`

8. `j && k;`

9. `message[2] - message[1];`

10. `message[0] - message[strlen(message)] - c;`

(6) [3]

What is the output of the following code fragment ?

```
char selector = 'x';
int out = 266;
while (1)
{
 switch (selector)
 {
  case 'x':
   printf("%x ", out);
  case 'y':
   printf("%d ", out);
  default:
   printf("default\n");
  case 'z':
   printf("%o ", out);
 }
 ++selector;
 out++;
 if (selector == 'z')
  break;
}
printf("\n%c\n", selector);
```

(7) [3]

Write code fragments whose behaviour is equivalent to the ones below, using only `while`

- ```
  for(index=0; index<MAX; index++)
   call1(index, 5);
  ```

- ```
  for(;;)
   call2(5);
  ```

(8) [6]

Write three functions, each with two arguments. The functions *change* an `unsigned long` integer given as a first argument. The second argument (an integer) indicates which bit is affected (counting from index 0, starting with the lowest order bit). The functions should test for a valid second argument for both LP32 and LP64 architectures. The first two functions do not return anything, the third returns TRUE or FALSE (define these appropriately).

1. `set()`: set the bit whose number is given by the second argument.

2. `invert()`: invert the bit whose number is given by the second argument.

3. `test()`: test if the bit whose number is given by the second argument is set.

(9) [4]

Given the function `tmp_name()` below,

```
char *tmp_name(void)
{
   char name[30];
   static int sequence = 0;

   strcpy(name, "tmp");
   name[3] = sequence + '0';
   name[4] = '\0';
   sequence = ++sequence % 10;

   return(name);
}
```

- Describe what is printed and why if `s = tmp_name();` is executed three times, followed by `printf("%s\n", s);`

`s` is declared as `char *`.

- How many unique strings can be produced by `tmp_name()` ?

(10) [2]

1. All variables in C are implicitly of what type ?

2. What is the largest number of typed in characters that can be read from `stdin` with `fgets(buffer, 10, stdin)` if `buffer` has been appropriately declared ?

(11) [8]

Assume a modular implementation of a stack data structure and operations (in `stack.c`) and its use (in `main.c`).

- Describe why you would declare the stack data structure `static` in `stack.c`.

- How can the stack data structure be accessed from other source files such as `main.c` if it is declared `static` in `stack.c` ?

- What would you put in `stack.h` if it were included in both `stack.c` and `main.c` ?

- Why is `stack.h` included in both files ?

(12) [4]

Give the type (in words) of the variables in the following declarations:

1. `char (*v)[20];`

2. `char *(v[20]);`

3.
```
struct node
{
 int (*get)(int);
 double size;
 struct node *p;
} n;
```

(13) [3]

Give `typedef`s (a series is allowed for clarity) for the following types:

1. Array of size 80 of pointers to a structure containing
    - an integer with name `i`,
    - a double named `d`, and
    - a pointer to this very structure.

2. Array of 20 arrays of 10 pointers to character.

(14) [6]

With the initial declarations

```
int array[]={10,20,30,40,50};
int value=0;
int *data_ptr=array; /* initially 0xbffff6c0 */
```

What are the values of the variables `value`, `data_ptr`, and `array` after each of the following statements (executed one after the other)

1. `value = *data_ptr++;`
    (a) `value`
    (b) `data_ptr`
    (c) `array`
2. `value = (*data_ptr)++;`
    (a) `value`
    (b) `data_ptr`
    (c) `array`
3. `value = *++data_ptr;`
    (a) `value`
    (b) `data_ptr`
    (c) `array`

4. `value = ++*data_ptr;`

   (a) `value`

   (b) `data_ptr`

   (c) `array`

(15) [8]

A program contains a polymorphic linked list. The elements of the linked list are either an integer or a character array of length 20 (but never both at the same time !).

- Give a `typedef` for the elements of the list.

- Write a function `print()` which takes a pointer to the first element as an argument and prints all integer elements of the list larger than 10.

  1. Iteratively.

2. Recursively.

(16) [2]

Given the following code fragment

```
typedef enum {RED, GREEN, BLUE} Colour;
Colour col=GREEN;
double values[3]={10.1, 20.2, 30.3};
```

What is the type and value of

- RED

- values[GREEN]

(17) [6]

Given the Makefile

```
CC= gcc
CFLAGS= -Wall -ansi -pedantic

test_stack: test_stack.o stack.o
        $(CC) $(CFLAGS) -o test_stack test_stack.o stack.o

calc: calc.o stack.o
        $(CC) $(CFLAGS) -o calc calc.o stack.o

test_stack.o: test_stack.c stack.h
        $(CC) $(CFLAGS) -c test_stack.c
```

```
calc.o: calc.c stack.h
	$(CC) $(CFLAGS) -c calc.c

stack.o: stack.c stack.h mydefs.h
	$(CC) $(CFLAGS) -c stack.c

stack.h: content.h

all: test_stack calc

clean:
	rm -f *.o

clean_target: clean
	rm -f test_stack calc
```

as well as the following result of `ls -l`

```
-rw-r--r--    1 hv       hv            482 Apr  2 02:41 Makefile
-rwxr-xr-x    1 hv       hv          25215 Apr  2 02:42 calc
-rw-r--r--    1 hv       hv           3013 Feb 27 15:43 calc.c
-rw-r--r--    1 hv       hv          13496 Apr  2 02:42 calc.o
-rw-rw-r--    1 hv       hv             90 Feb  8 18:00 content.h
-rw-rw-r--    1 hv       hv            149 Apr  2 02:44 mydefs.h
-rw-rw-r--    1 hv       hv           2707 Feb 27 15:43 stack.c
-rw-rw-r--    1 hv       hv            421 Apr  2 02:42 stack.h
-rw-r--r--    1 hv       hv          10824 Apr  2 02:42 stack.o
-rwxr-xr-x    1 hv       hv          23219 Apr  2 02:42 test_stack
-rw-r--r--    1 hv       hv           1497 Feb 13 13:45 test_stack.c
-rw-r--r--    1 hv       hv          10324 Apr  2 02:42 test_stack.o
```

1. Which commands will be executed by `make` ?

2. Which files will remain after `make clean_target` ?

3. Which targets are produced by `make all` after `make clean_target` ?

(18) [2]

Explain the use of suffix rules in makefiles. As an example, show how to specify the transformation of C source files to object files.

(19) [2]

What is the purpose of the `where` command in `gdb` ?

(20) [3]

Explain the different phases of the (gcc) C compiler (by preference with a figure).

(21) [6]

1. Describe how command line arguments are passed to a program.

2. Describe (the essence of) the use of the `getopt()` library function.

(22) [4]

Which types of information can be obtained using a profiler ?