Exam in EDAA25 C Programming

October 30, 2014, 14-19

Inga hjälpmedel!

Examinator: Jonas Skeppstedt

Grading instructions

- In general: assess if a function or program works as intended while ignoring syntax errors. That gives full score. Then subtract each syntax error from that score and give at least zero points.
- If the code doesn't work, judge youself on how serious error it contains, keeping in mind the obvious that the intent is to give the proper grade.
- If a student makes the same mistake n times, the student's score will be reduced n times.
- In general -1 per trivial syntax error which can be regarded as a typo.
- In general -2 per serious syntax error which indicates the student needs more practising.

30 out of 60p are needed to pass the exam. You are not required to follow any particular coding style but your program should of course be readable. If you call a function which may fail, such as `malloc`, you must check whether the call succeeded or not.

1. (10p) Implement the function

   ```
   #include <string.h>
   char* strcat(char* restrict dest, const char* restrict src);
   ```

   appends the string pointed to by `src` to the end of the string pointed to by `dest`, and the first character appended overwrites the null character of `dest`. The strings must not overlap, but if they do the behavior is undefined. The return value is `dest`.

   Hint: when you implement the function, the presence of `restrict` can be completely ignored since it only affects user of the function and not the implementation.

2. (3p) In which sense is the function in the previous question insecure (not counting the fact that the strings are not allowed to overlap as a security issue)?

3. (2p) What does the following function print, and why?

   ```
   int main(void)
   {
           char    b[] = "Arrays and pointers in C are not the same thing.";
           char*   c = b;

           if (sizeof b == sizeof c)
                   printf("THEN\n");
           else
                   printf("ELSE\n");

           return 0;
   }
   ```

4. (5p) Consider the previous question again but now with the following variant.

```
void f(const char b[100], const char* c)
{
        if (sizeof b == sizeof c)
                printf("THEN\n");
        else
                printf("ELSE\n");
}

int main(void)
{
        f("Arrays", " and pointers.");

        return 0;
}
```

   What is the output this time and why?

5. (10p) Implement a stack that can be used in an RPN-calculator such as the one in Assignment 1 from September. Your stack should have `int` as the type of the stored data, and it must be implemented with a heap allocation for each time you push, and deallocation each time something is popped. Your implementation should check that it was possible to allocate memory and that no attempt is made to pop from an empty stack. At any error, call `exit(1)`.

   In addition to `push` and `pop`, you should implement a function `new_stack` to create an empty stack (which, of course, may simply return a null pointer if you wish that), and `free_stack` to deallocate any memory used by the stack. These functions should be used as in the code below, and the return value from `main` should be zero (which it will be if `free_stack` sets what the parameter points at to `NULL`).

```
int main(void)
{
        stack_t*        stack;
        int             a;

        stack = new_stack();

        push(stack, 2014);
        a = pop(stack);
        push(stack, 10);
        push(stack, 30);
        free_stack(&stack);

        return stack == NULL ? 0 : 1;
}
```

6. (20p) Give brief descriptions of each of the following.

   - `->`
   - `?:`
   - `:0`
   - `...`
   - `volatile`
   - Function scope

- Internal linkage
- ANSI C aliasing rules
- Flexible array member
- Variable length array

7. (5p) Use the declaration of a single linked list below, and implement the function `reverse`, which should reverse the list and set what the parameter points at to the new first list element.

   Your implementation is, however, not allowed to use neither `goto` or any loop syntax (i.e. `for`, `while`, and `do-while` are forbidden). Instead, obviously, use recursion. Furthermore you are allowed to visit each list node only once. You are allowed to use additional functions (which may have any number of parameters). Except for a constant number of local variables (such as a few pointers), you are not allowed to allocate memory either from the heap nor from the stack (which you might consider for a variable length array but that is forbidden).

```
typedef struct list_t list_t;

struct list_t {
        list_t* next;
        void*   data;
};

void reverse(list_t** list);
```

8. (5p) Implement the macro `isnan(a)` which returns a non-zero value if the floating point type parameter is a so called not-a-number, `NaN`. Hint: you do **not** need to know how a `NaN` value is represented — it's possible (and easier in some sense) to solve the problem without using that information, but if you do know it, you are allowed to use that knowledge and check the representation, but do not break any rule of the C standard!