Programming and Data Structures in C
Laboratory instruction

March 7, 2025

General rules

When writing programs follow these rules:

- Clearly divide program into functions responsible for particular tasks.
- Use proper naming:
  - names of functions must be given reasonably to reflect their responsibility (what they do),
  - names of variables must be given reasonably to reflect their meaning (what they are),
  - there is nothing wrong in using longer names of functions and variables composed from a few words,
  - choose naming convention you like and use it consequently in your programs,
  - do not mix different languages, e.g. Polish and English.
- Keep functions short.
- Do not duplicate code.
- Try to avoid comments. Instead use self-explaining names for functions and variables. However, put a comment in case where the name itself is not sufficient to explain well what the function does or what the variable means. When describing variables and constants use substantival form, e.g.: ,,angle of rotation in radians" instead of ,,variable used to represent angle of rotation in radians".
- Using ,,magic numbers" is not allowed. All constants used in the program must be defined using the **#define** preprocessor directive.
- Programs must be compiled using −g, −Wall and −pedantic options e.g.:

```
gcc -g -Wall -pedantic myprog.c -o myprog
```

Tasks

1 Rotating polygon

Time for writing program: 1 week.

Task description

Develop program that displays rotating polygon. Use functions from the SGL library (Simple Graphics Library[1]). Initial values of the following variables must be defined using the **#define** directive:

- number of vertices,
- angle of rotation between consecutive frames.

Guidelines

- The polygon should rotate smoothly, without glitching and grow or shrink in each frame.
- Make sure that variable which represents angle of rotation never overflows. To achieve this you can keep the variable in the range of $\langle 0, 2\pi \rangle$.
- Clear screen before drawing every new frame.
- The example of the program is available on the web-page for the PDSC lecture[2].
- If your program runs into infinite loop during development enter the following command and click on the window of your program to close it:

  ```
  xkill
  ```

2 Tower of Hanoi

Time for writing program: 2 weeks.

Task description

The aim of the exercise is to write a well-known Tower of Hanoi[3] game. The array that stores images of stacks must be equal to number of discs and pegs. Do not use bigger array than required.

[1] `http://neo.dmcs.p.lodz.pl/pdsc/graphmanual.html`
[2] `http://neo.dmcs.p.lodz.pl/pdsc/rotpoly.html`
[3] `http://en.wikipedia.org/wiki/Tower_of_Hanoi`

Guidelines

- Enable UndefinedBehaviorSanitizer to check boundary errors. To turn on UndefinedBehavior-Sanitizer in projects that use the SGL library modify the makefile as follows:
  - replace ,,gcc" with ,,gcc -fsanitize=undefined" (change required in two places).
- The moving discs must be animated.
- All parameters should be parametrized, e.g. when you change the number of discs or pegs all dimensions must be recalculated to obtain nice view on the screen. Incorrect dimensions of discs are not allowed.
- The game should allow displaying up to 10 pegs and large number of discs.
- Divide the program into intuitive functions.
- Use correct functions for reading key for pieces movement, the game must be smart, jumping and freezing of discs are not allowed when you move up, left, right or down.
- Do not allocate memory, use static arrays.
- Use the following keys to control the game: 1, 2 ,..., 0, escape, enter.
- Display nice ,,Congratulation" or ,,Try again" messages when game is finished (escape or enter).
- The example of the game is available on the web-page for the PDSC lecture[4].

3 Tetris

Time for writing program: 2 weeks.

Task description

The aim of the exercise is to write a well-known Tetris[5] game. We supply you a definition of 4x4 pieces. Please use a rectangular matrix as a field for the game, set its initial size to 10x20. Do not use in your program bigger arrays than required.

Guidelines

- Enable UndefinedBehaviorSanitizer to check boundary errors.
- The falling pieces must be animated.
- All parameters should be parametrized, e.g. when you change the game field array to 20x30.
- Divide the program into intuitive functions.
- The game must be responsive in the time between falling of the pieces. It should be possible to perform a move at any time and the result should be visible immediately.
- Use correct functions for reading key for pieces movement, the game must be smart, jumping and freezing of pieces are not allowed when you move left, right or rotate the piece.
- Do not allocate memory, use static arrays.

[4] http://neo.dmcs.p.lodz.pl/pdsc/hanoi.html
[5] http://en.wikipedia.org/wiki/Tetris

- Use the following keys to control the game: left arrow, right arrow, down arrow (falling down), space (rotation), enter and escape.
- Display nice ,,Congratulation" or ,,Try again" messages when the game is finished (escape or enter).
- The example of the game and pieces definition are available on the web-page for the PDSC lecture[6] [7].

4 Text processing

Time for writing program: 2 weeks.

Task description

Write the program which processes text read from the standard input stream and writes the processed text to the standard output stream. The input text contains several lines, some of them can be empty. Each non empty line contains a positive binary number expressed as a sequence of '0' and '1' characters, with possible leading and/or trailing whitespace characters. The processing computes sum of all numbers and prints it. After that the numbers are printed without whitespace characters. Neither maximum line length nor maximum number of lines is defined, so fixed-size buffers are not acceptable.

If the format of the input is incorrect, e.g. it contains any other characters than whitespace or digits, print an error message.

For example, for the following input text:

```
      1000100111


  101001001010100
00001111001
```

the expected output text is:

```
Sum:
101010011110100

Input numbers:
1000100111
101001001010100
00001111001
```

Guidelines

- Maximum line length is not defined, so you will need to use dynamic memory allocation. Define the following function:

  ```
  char* getLine()
  ```

  which reads entire line from the standard input stream character by character using the `getchar` function and stores it in a dynamically allocated array. Expand the array as needed. Return the pointer to the beginning of the line or NULL, if the *end-of-file* has been encountered before any character has been read. Beware of the final line not ending with the '\n' character.

[6] `http://neo.dmcs.p.lodz.pl/pdsc/tetris.html`
[7] `http://neo.dmcs.p.lodz.pl/pdsc/pieces.inl`

- Do not read the text from a file, use only the standard input.
- You can still test your program using a text file. To redirect the standard input stream invoke your program in the following way:

  ```
  ./program_name < input_filename
  ```

- The addition algorithm must operate on characters and strings and can not convert input numbers to integers.
- Add only two input numbers in a single step of the addition algorithm.
- Make sure your program works correctly when the input text is empty.
- Handle all possible memory allocation errors. Use the library for random injection of memory allocation errors available on the web-page for the PDSC lecture[8]. Test your program against different rates of random error injection.
- Use valgrind to verify there are no memory leaks.

5 strtol function

Time for writing program: 1 week.

Task description

Implement the `strtol` function which is available in the standard library[9]. Declaration of the function is as follows:

```
long int strtol (const char* nptr, char** endptr, int base)
```

The function converts the initial part of the string in `nptr` to a long integer value according to the given base, which must be between 2 and 36 inclusive, or be the special value 0.

Guidelines

- Correctly check if the converted number is not overflowed.
- The function must process correctly negative numbers in each step of the conversion. It is not allowed to process modulo number and modify its sign in the last step of the conversion.
- Carefully check if the program does not read chars outside the table pointed by `nptr`.
- Use the test program provided on the web-page for the PDSC lecture[10].

6 Banking system

Time for writing program: 2 weeks.

[8] `http://neo.dmcs.p.lodz.pl/pdsc/rand_malloc.tgz`
[9] `https://en.cppreference.com/w/c/string/byte/strtol`
[10] `http://neo.dmcs.p.lodz.pl/pdsc/strtol.tgz`

Task description

The aim of the exercise is to write a program that simulates a customer bank account management system. The system should allow to carry out basic operations on your virtual bank accounts. The system should allow to:

- create a new account with the following fields describing the customer:
  - account number (generated by the program),
  - name,
  - surname,
  - address,
  - identification number (PESEL),
  - current balance of regular account,
  - current balance of bank loan,
- list all accounts present in the database,
- search for an account using one of the following fields: account number, name, surname, address, identification number,
- make a deposit,
- make a withdrawal,
- make a money transfer between two accounts,
- take a loan and transfer money to account (interest rate is entered when the loan is taken),
- pay the debt with money from account.

Guidelines

- All data should be read from and displayed on the terminal.
- The records must be read from the file and updated in the file, caching the records in memory is not allowed. Update must take place after every modifying operation
- The program must accept unlimited number of accounts
- Always verify if a correct string of data was read from the terminal before you process the data.
- Limit numbers and strings to avoid overflow and memory errors (boundary errors).
- Check if current operation is allowed.
- Ask for confirmation before every modifying operation on the bank account. Asking for confirmation of every operation is not accepted.
- Do not use account identifiers longer than a few characters if typing them would be necessary to perform operations

7 bsearch function

Time for writing program: 1 week.

Task description

Implement the `bsearch` function - binary search of a sorted array - which is available in the standard library[11]. Declaration of the function is as follows:

```
void* bsearch(const void* key, const void* base,
              size_t num, size_t size,
              int (*compar)(const void*, const void*));
```

The function searches an array of num objects, the initial member of which is pointed to by `base`, for a member that matches the object pointed to by `key`. The size of each member of the array is specified by `size`. The contents of the array should be in ascending sorted order according to the comparison function referenced by `compar`. The `compar` routine is expected to have two arguments which point to the key object and to an array member, in that order, and should return an integer less than, equal to, or greater than zero if the key object is found, respectively, to be less than, to match, or be greater than the array member. The `bsearch` function returns a pointer to a matching member of the array, or NULL if no match is found. If there are multiple elements that match the key, the element returned is unspecified.

[11] `https://en.cppreference.com/w/c/algorithm/bsearch`