Programming and Data Structures in C
Instruction for students

Adam Piotrowski Dariusz Makowski Wojciech Sankowski

11 kwietnia 2016

General rules

When writing programs please note that:

- Program must be clearly divided into functions responsible for particular tasks.
- 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.
- You should choose naming convention you like and use it consequently in your programs. Do not mix different languages, e.g. Polish with English.
- 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" instead of „variable used to represent angle of rotation".
- Using „magic numbers" is not allowed. All constants used in the program must be defined using the „#define" preprocessor directive.
- Kate[1] is the suggested text editor.
- Programs must be compiled using -g, -Wall, -pedantic, -Werror and -std=c99 options e.g.: gcc -g -Wall -pedantic -Werror -std=c99 myprog.c -o myprog.out

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[2]). Initial values of the following variables must be defined using the „#define" directive:

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

[1] http://www.kate-editor.org

[2] link to the library is available on the web-page for the PDSC lecture http://neo.dmcs.p.lodz.pl/pdsc/

Guidelines

- Make sure that variable which represents angle of rotation never exceeds 2π.
- Clear screen before drawing every new frame.
- The example of the program is available on the web-page for the PDSC lecture. To run the program download it and change file permissions (Properties - Permissions - Allow executing file as program). To close the program enter the following command and click on the window of the rotating polygon program:
  xkill

Tower of Hanoi

Time for writing program: 2 weeks.

Task description

The aim of the exercise is to write a well-known 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.

Guidelines

- If you work under Ubuntu 14.04 or newer, turn on AddressSanitizer to check boundary errors. To introduce AddressSanitizer in projects that use SGL library modify the makefile as follows:
  - Replace „gcc" with „gcc -fsanitize=address" (change required in two places).
- If you work under Ubuntu 11.04, turn on fbounds-checking and gcc-4.0.2 compiler to check boundary errors. To introduce fbounds-checking in projects that use SGL library modify the makefile as follows:
  - Replace „gcc" with „/usr/local/gcc-4.0.2/bin/gcc -fbounds-checking" (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.
- Divide program in 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 left, right or up
- Do not allocate memory, use static arrays.
- Use 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. To run the program download it and change file permissions (Properties - Permissions - Allow executing file as program).

[3] http://en.wikipedia.org/wiki/Tower_of_Hanoi

Tetris

Time for writing program: 2 weeks.

Task description

The aim of the exercise is to write a well-known Tetris[4] game. We supply you a definition of 4x4 pieces. Please use a defined square matrix as a field for Tetris game, initial values 10x20. Do not use bigger array than required.

Guidelines

- Use AddressSanitizer or fbounds-checking and gcc-4.0.2 compiler to check boundary errors.
- The falling pieces must be animated.
- All parameters should be parametrized, e.g. when you change the field array of the Tetris game to 20x30.
- Divide program in intuitive functions.
- 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 piece.
- Do not allocate memory, use static arrays.
- Use following keys to control the game left, right, down arrows, space, enter and escape.
- Display nice „Congratulation" or „Try again" messages when game is finished (escape or enter).
- The example of the game and pieces definition are available on the web-page for the PDSC lecture. To run the program download it and change file permissions (Properties - Permissions - Allow executing file as program).

Determinant

Time for writing program: 2 weeks.

Task description

The aim of the program is to calculate a determinant of square matrix using Laplace recursive formula.

$$det(A) = \sum_{j=1}^{n} (-1)^{1+j} \cdot a_{1j} \cdot A_{1j}, \ where$$

A_{1j} is the minor matrix with removed the first row and j-th column.

Guidelines

- Use malloc function for dynamic memory allocation for the minor matrix.
- Check if malloc correctly allocated memory; use valgrind to debug memory leakage.
- Check scanf return value and read the input string in a safe way.
- Check if matrix is rectangular.
- Check order of matrix, for 1 or 2 it is not necessary to calculate determinant.
- Perform tests of determinants for matrices with order between 3 and 7 [5].

[4] http://en.wikipedia.org/wiki/Tetris

[5] The test script and matrices are available on the web-page for the PDSC lecture http://neo.dmcs.p.lodz.pl/pdsc/

strtol function

Time for writing program: 2 weeks.

Task description

Implement a strtol() function that 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.

```
#include <stdlib.h>
val=(long int)strtol((const char *)nptr, (char **)endptr, (int)base)
```

The strtol() function is available in C standard library. The detailed description of the function is available with Linux manual (man 3 strtol).

Guidelines

- Correctly check if the converted number is not overflowed .
- The function must process correctly negative numbers in each step of the convention. It is not allowed to process modulo number and modify its sign in the last step of the convention.
- Carefully check if program do not read chars outside the table pointed by **nptr**.
- Use the test program provided on the web-page for the PDSC lecture [6].

Banking system

Time for writing program: 2 weeks.

Task description

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

The system should allow to:

- Create a new account.
- Display information about all account present in the database.
- Find data using all of the available fields (Account Number, Name and Surname, Owner Address, Identification Number, Balance).
- Make a money transfer.
- Make a payment.
- Make a withdrawal.

The system should support the following data describing customers:

- Account Number.
- Name and Surname of the account owner.
- Address of the account owner.
- Owner Identification Number (PESEL).
- Current Balance.

[6] http://neo.dmcs.p.lodz.pl/pdsc/

Guidelines

- All data should be read and displayed on the terminal.
- Please always verify if a correct string of data was read from the terminal before you process the data.
- Check if current operation is allowed.
- Limit numbers and strings to avoid overflow and memory errors (boundary errors)
- The database must use file, it is not allowed to allocate memory for storing records.
- Every operation on the bank account must be confirmed.

bsearch function

Time for writing program: 1 week.

Task description

Implement bsearch function - binary search of a sorted array. Declaration of function is presented below.

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

The bsearch() function searches an array of nmemb 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.[7]

Linked list

Time for writing program: 2 weeks.

Task description

Design and implement unidirectional linked list of pointers to null-terminated strings. The complete list interface must include following functions:

- init – initializes empty list, returns pointer to newly created list,
- length – returns number of elements stores on the list identified by parameter,
- display – shows all elements stored on the list identified by a parameter,
- push – inserts new element at the beginning on the list identified by parameter,
- pop – removes first element from the end on the list identified by parameter and return stored value,

[7] source: linux.die.net/man/3/bsearch

- destroy – destroys the list identified by parameter,
- append – inserts new element at the end of the list identified by parameter,
- copy – creates copy of the list identified by parameter,
- reverse – reverses order of elements of the list identified by parameter,
- sort – sort elements of the list identified by parameter

Static memory allocation is not allowed, memory for list nodes and stored null-terminated strings must be allocated dynamically.