

Algorithms 2018

Lab 01 (8%)

Topics: Pointers in C

(Reading: Stroustrup “Programming Principles and Practices”. Chapters 17, 18.)

Problem 01 (1%)

(pointer declaration, initialization and dereference)

You have to do following operations in a simple C-program:

- create 2 local integer variables x, y.
- create local pointer to integer p.
- assign value 1 to variable x.
- assign address of variable x to pointer p.
- print value of address of variable x using pointer p.
- print value of variable x using the variable x itself.
- print value of variable x using pointer p.
- assign value 2 to variable x using pointer p.
- print new value of x.
- assign the square of value in variable x to variable y using pointer p instead of variable x.
- print value of variable y.

Problem 02 (1%)

(importance of pointers in C: pass parameters)

Write a simple C-program which has two functions to swap values of integer variables: badSwap(int, int), goodSwap(int*, int*). Program has to read two integer values from standard input into variables x and y; try to swap values of x and y using badSwap; try to swap values of x and y using goodSwap; before and after each try program has to print values of x and y. Explain why it's necessary to use pointers in goodSwap and scanf functions.

Problem 03 (1%)

(importance of pointers in C: arrays and pointers)

Write a simple C-program which

- reads N integer numbers into an array of integers;
- computes the sum of all elements using subscript operator;
- computes the sum of all elements using pointer arithmetic;

Explain why pointer arithmetic might be more preferable to work with arrays in C.

Problem 04 (1%)

(importance of pointers in C: pass arrays to functions)

Write a simple C-program which

- initializes array a by N integer numbers;
- initializes array b by M integer numbers;
- prints the contents of a using function printArray
- prints the contents of b using function printArray

Implement function printArray to print any integer array of any length; use pointer arithmetic to print each value of array.

Problem 05 (1%)

(global memory, stack memory, dynamic memory)

You have to do following operations in a simple C-program:

- define global array of integers; explain the lifetime, scope and initialization of global variables;
- define local array of integers in function main; explain the lifetime, scope and initialization of local variables;
- define local array of integers in function f and print its first and second element; call function f in main function; explain the lifetime, scope and initialization of local variables;
- define function `int* createDynArray(int size, int initValue)` which creates array of given size in dynamic memory using `malloc` function and returns pointer to it;
- in function main read the size and `initValue` from standard input; create a dynamic array of that size filled by `initValue` using function `createDynArray(size, initValue)`
- in function main print the first and second elements of dynamic array using index operator;
- release memory of dynamic array using function `free`;

Explain the difference between global, local and dynamic memory.

Problem 06 (1%)

(dynamic arrays)

Write simple C-program using dynamic memory. This program has to

- read the size of array and its elements from standard input;
- print the contents of that array;
- reverse array using pointer arithmetic;
- print its elements again.

Problem 07 (1%)

(dynamic arrays)

Write simple C-program to read arbitrary amount of integer numbers from standard input (stop when user enters Ctrl-Z); reverse the entered sequence; print them;

Problem 08 (1%)

(how we can create something like C++ `std::vector` in C programs)

Solve previous problem using structure:

- `VecInt { int *data; int size; int capacity; };`

and functions:

- `void init(struct VecInt* this);`
- `void pushBack(struct VecInt* this, int e);`
- `void destroy(struct VecInt* this);`