Laboratory 1

Memory management, pointers

based on Instruction 1 for “Programowanie w języku C 2” by mgr inż. Leszek Ciopiński

---

Pointers

The C language provides dynamic allocation and dealocation of RAM. In order to access the allocated part of the memory, its starting point address is used, which is stored in a so called Pointer. Because the address informs only where to search in a memory for the data, but not how big it is, each pointer should have a type which tells how many bytes are dedicated to the data. The pointer declaration is as follows:

int* ptr;

The declaration in such a form does not store any information and does not allocate memory for further usage.
In order to assign an address of existing variable to the pointer, the ‘&’ should be used, to read the value assigned to the place where points the pointer, ‘*’ should be used:

```
int varInt=5;
int* ptr;
ptr = &varInt;
printf(“Value: %d”, *ptr);
```

Memory allocation

There are two commonly used functions to allocate a memory: malloc and calloc.

The malloc () function is the basic function for dynamic memory allocation in C. Description of the functions is presented below:

```
#include <stdlib.h>
void *malloc(size_t size);
```

As a parameter, the function assumes the size of the area to be allocated, expressed in bytes. In practice, the sizeof () function is most often used to determine the size. The value returned by malloc()

on success is a void pointer. Therefore, this value should be cast to obtain the appropriate type. In the event that it was impossible to reserve a sufficient amount of memory, NULL is returned.

The sizeof () function takes any variable or type name as a parameter. The return value is the memory size occupied by the given variable or type in bytes. For pointers, this function always returns the same size, regardless of the size of allocated memory and whether the memory has already been allocated or not. Examples:

```
int  a, b, c;
a =  sizeof(int);
b =  sizeof(c);
//a ==b
```

The calloc() function is mainly used to allocate arrays. Its description is as follows:

```
#include <stdlib.h>
void *calloc(size_t nelem, size_t elsize);
```

The nelem parameter specifies how many elements the array is to be, and the elsize parameter is the size in bytes of each element. Unlike the malloc () function, the calloc () function also fills all its bits with the value 0 after reserving memory. Similarly to the malloc () function, if the returned value is NULL, it means that the operation failed. Otherwise, the returned pointer should be cast to the appropriate data type.

Freeing the memory

The basic way to free up previously reserved memory is to use the free () function, whose description is as follows:

```
#include <stdlib.h>

void free(void *ptr);
```

The only parameter passed to the function is the pointer to the freed memory area. Please note that you must not free an unreserved area of memory, as program operation is not specified in such cases.

An example:

```
#define ROZMIAR 10

int i;

int **tabliczka = malloc(ROZMIAR * sizeof *tabliczka);

*tabliczka = malloc(ROZMIAR * ROZMIAR * sizeof **tabliczka);
```

```
for (i = 1; i<ROZMIAR; ++i) {
     tabliczka[i] = tabliczka[0] + (i * ROZMIAR);
}
for (i = 0; i<ROZMIAR; ++i) {
     int j;
     for (j = 0; j<ROZMIAR; ++j) {
          tabliczka[i][j] = (i+1)*(j+1);
     }
}
free(*tabliczka);
free(tabliczka);
```

(source: pl.wikibooks.org)

Tasks:

1. Declare an int type variable and assign a value to it. Then create a pointer to this variable and use it to display the value previously entered into the statically declared variable.

2. Create a dynamic array of numbers. The program should firstly ask for the number of elements and then for each number. As a result, the program should write sorted array.