Homework 1

1. Implement the function `void v_alloc_table_add_5 (int iSize)`, which allocates in dynamic way memory for one-dimensional array of int variables. The size of an array is specified by the (iSize) parameter. The array elements should be initiated on offset+5.
   - When array is allocated and initialized display all array elements.
   - Remember that you need to deallocate memory using delete before end of the „program".
   - The function should be protected against an invalid iSize parameter value.
   - Finally, the question, should the value 5 appear directly in the code of `v_alloc_table_add_5` function? Discuss it.

2. Implement the function
`bool b_alloc_table_2_dim (int ??? piTable, int iSizeX, int iSizeY)` for which:

   - The allocation should be made in such way that after execution of the following code

```
int ** pi_table;
b_alloc_table_2_dim (??? pi_table, 5, 3)
```

   pi_table should pointed an int array of 5 * 3 size.

   - If the operation succeeds, the function should return true and false otherwise.
   - Determine what to insert instead of ??? when the reference cannot be used.

3. Implement the function
`bool b_dealloc_table_2_dim (int ??? piTable, int iSizeX, int iSizeY)`; which
   - Deallocates a two-dimensional array of int type.
   - If the operation succeeds, the function should return true and false otherwise.
   - Determine what to insert instead of ??? when the reference cannot be used.
   - Will there be a difference compared with function `b_alloc_table_2_dim`?
   - Can `b_dealloc_table_2_dim` have fewer parameters then function `b_alloc_table_2_dim`?

4. Define „CTable" class with the following constructors:

   - Without parameters CTable ()
     - Aassigns a default value to the s_name field (remember to use appropriate constants values).
     - Display on the screen the following text: „without: '<s_name>' ", where <s_name> is the value of the s name field.
     - Creates an array of int with the default length.
   - With the parameter: CTable (string sName, int iTableLen)
     - Aassigning the field s_name, the value of sName
     - Display on the screen the following text: „parameter: '<s_ name>' "

- Assigns an array length equal to iTableLen

- copying: CTable (CTable & pcOther)
  - assigning the field s name, the value pcOther.s name and appending the text „_copy”. For example, if pcOther.s_name = "test" then the value of the s name field for the object created by the copy constructor would be: „test_copy”
  - Copies the array int
  - Display on the screen the text: „copy: ‘<s_name>’ ”

Additionally the class should have:

- Destructor, displaying the following text on the screen: „removing: ‘<s_name>’ ”
- The method, void vSetName (string sName), assigning the value sName to the field s_name
- The method, bool bSetNewSize (int iTableLen), changing the length of the array and returning information whether it was possible or not.
- The method, CTable * pcClone (), which returns a new object of the CTable class, which is a clone of the object for which pcClone () was called. E.g:

For example:

```
CTable c_tab;
CTable * pc_new_tab;
pc_new_tab = c_tab.pcClone ();
```

Consider whether the pcClone () method is functionally similar to other elements of the code? If so, how do you write the program to take advantage of this fact and not copy the code unnecessarily?

Write two procedures:

```
void  v_mod_tab(CTable  *pcTab, int iNewSize);
void  v_mod_tab(CTable  cTab, int iNewSize);
```

Check which procedure will modify the object given as the parameter. Will any copies will be created?

Test static and dynamic allocation of CTable objects. When is the constructor called and when the destructor is called? Check the constructor and destructors call when allocating an array of CTable objects.