Exercise #3	Algorithms and Data Structures	
	Topic: Dynamic data structures – lists	Version: 1.0 / 2019
	Prepared by: dr inż. Grzegorz Łukawski & dr inż. Barbara Łukawska	

1) Dynamic data structures

1.1) The definition

Dynamic data structures – data structures implemented using pointers (not references). The size of memory occupied by such structure is allocated dynamically if needed.

Main features of lists:

- Its size is unrestricted, it can store initially unknown and variable number of elements.
- Elements of a list are linked with pointers in such a way that the previous element points to the next one.
- An element of a list is a structure ("struct"), storing a piece of data and an additional pointer to the next element of the same type.
- Pointer to the first element of a list is called the root.
- The last element of a list holds a null pointer.

1.2) Types of lists

- **Unidirectional list:** Every element of a list stores a single pointer, end of the list is marked with a null pointer. Can be accessed sequentially from the first to the last element (only).
- **Bidirectional list:** Every element stores two pointers. The extra pointer points the previous element of a list. Can be accessed from the first to the last and from the last to the first element.
- **Ordered list:** All elements of a list are always ordered using some key (ascending or descending).
- **Cyclic list:** Last element of a list is linked to the first element of the list. There is no null pointer anywhere in the list!

1.3) Example implementation of a unidirectional list

Data structure:

```
struct element {
	string name;
	float price;
	struct element *next;
};
```

Pointer to the first element of the list (root):

```
struct element *list = NULL;
```

Adding an element to the beginning of the list:

```
void AddToList(struct element **list, string name, float price) {
	struct element *p = new element;
	p->name = name;
	p->price = price;
	p->next = *list;
	*list = p;
}
```

Example usage:

```
string name;
cout << "NAME: ";
cin >> name;
float price;
cout << "PRICE: ";
cin >> price;
AddToList(&list, name, price);
```

Displaying the complete list:

```
void DisplayList(struct element **list) {
	struct element *p = *list;
	while (p != NULL) {
		cout << p->name << " : " << p->price << endl;
		p = p->next;
	}
}
```

Example usage:

```
DisplayList(&list);
```

Deleting the list and releasing the memory:

```
void DeleteList(struct element **list) {
	struct element *current;
	struct element *p = *list;
	while (p != NULL) {
		current = p;
		p = p->next;
		delete current;
	}
	*list = NULL;
}
```

Example usage:

```
DeleteList(&list);
```

---

2) Exercises

Write a program in **C++**, where the user can add new elements to an unordered list and display its content. Remember to delete the list before the end of the program! Sample code shown above may be used for the implementation. Expand your program with the following features:

A) Create a function for searching the list for a price of a given “item”. If user enters a name which is not found on the list, display proper message.

B) Modify the list, so elements on the list are ordered by price. New items added to the list should be inserted into the right place (the beginning, the end or somewhere inside the list).

Additional exercises:

C) Turn the list into a cyclic list, modify all the functions respectively. Remember that there is no NULL pointer in a cyclic list!

D) Turn the list into a bidirectional list, modify all the functions respectively.