Reducing the Amount of Hard Drive Memory Used For Storing Elements of Abstract Data Types

Grigoriy Pronin[1], Olga Popova[2]

Department of Information Systems and Programming, Kuban State Technological University, Russia

***Abstract*— *This article proposes, presents and analyzes a new method of saving hard drive memory, used by Abstract Data Types. Abstract Data Types represents an important topic across a variety of application domains, but for several projects the method of memory allocation for the use of Abstract Data Types, presented in the built-in classes of the programming language C#, is not suitable by the reason oflarge possible memory loss. Method, reviewed in this article, can be implemented in other object-oriented programming languages. Sphere of application of represented method encompasses multiple types of Abstract Data Types, such as "list", "stack", "queue", "deq" and several other. Implementation of reviewed solution requires usage of built-in class "array". Realisated classes become alternative patterns for built-in classes. The ways of usage of a new solutions is similary to the previous ones. Implementation of reviewed solution is done using Microsoft Visual Studio 2017.The results include theoretical conclusions drawn on the basis of practical research.*

***Keywords*— *Visual Studio 2017, Hard Drive Memory, Method, Abstract Data Type*.**

I. INTRODUCTION

This article discusses the concept of abstract data types (ADT). Since the concept was formulated in 1974, abstract data types became an important part of the programming theory. Currently, the concept of ADT is one of the most popular methods of programming, along with the object-oriented programming.

1.1 Features of ADT

A distinctive feature of abstract data types is the fact that the functionality of the program component can be implemented in various ways. Different implementations of abstract data types are interchangeable.

"ATD is described by the mathematical theory of algebraic systems. Operations are implemented as functions of one or more parameters. A description of the operations, including a description of the argument types and return values, is called an algebraic system signature. Signatures represent a mathematical model of an abstract data type. This circumstance makes it possible to describe the functions of the program, specified by means of ATD, as algebraic systems" [1].

1.2 ADT in C#

"The C# programming language has many constructs that was borrowed from C++ and Java programming languages. However, it includes a number of new designs. Like the Java language, all instances of C# classes are dynamic objects and are hosted on a "heap".

For all classes of predefined standard constructors, which provide the initial values of data. It is possible to add a number of new constructors that provide specific initialization of data elements, and any instance variable that is not initialized by an abstract constructor will receive a value from the standard constructor.

Destructors are also provided in C#, but they are rarely used because automatic garbage collection is used.

Structs or classes can be used to create ATDs in C#. The structures in C# differ significantly from their counterparts in C++. They can have constructors, properties, methods, data fields, and can implement interfaces but do not support inheritance.

A significant difference between structures and classes is that structures are value types and classes are reference types. Structures are placed on the stack, not on the heap. When used as parameters, structures are passed by value. All types-values in C#, including all primitive types, belong to structures. Instances of structures can be created by declaring them, like instances of other predefined value types, such as int or float. They can also be created using the new operator, which calls the constructor to initialize them." [2]

Structures are used to implement relatively small and simple types that will never be inherited. They are useful for creating objects that are placed on the stack (as opposed to the heap).

Three levels of access (public, private, and protected) in C# are specified by the same path as Java's.

1.3 Built-in ADT classes of C#

"Microsoft Visual Studio's C# has several built-in classes for realisation of different kinds of ADT. All the classes have the same structure, divided by the types of ADT. The difference of classes is in basic methods of their own, but the way of creation of a new variable for ADT's is the same: firstly, variable is created with only one element cell in it, but if program will identify this cell as filled cell, it will extend variable to the size that is equal to the previous one multiplied by 2" [3].

This method of variable creation has one big negative side: for example, if program usess a variable with 128 cells in it, built-in class of C# will extend this variable to the size of 256, because built-in class was created with a feature of saving extra cells for a new possible elements.

Creation extra cells is important, but even empty ones require memory to be stored, and in that case using built-in classes may lead to a loss of storage memory.

II. METHOD

The next method have been implemented on C# programming language for class "stack" but the main idea of it can be used faor any ADT on ane programming language.

In this program, a class describing the stack was created manually. The beginning of the MyStack class shows that this class was implemented and developed on the basis of the built-in collection System.Array

```
class MyStack<T>
 {
T[] arr;
public MyStack()
 {
 arr = newT[0];
}
```

The Push method initially increases the stack size by 1, and then adds the specified element to the end of the array that the stack consists of:

```
public void Push(T item)
 {
Array.Resize(ref arr, arr.Length + 1);
arr[arr.Length - 1] = item;
 }
```

The Pop method is implemented by saving the last element of the stack and then reducing the stack size by 1:

```
public T Pop()
 {
 T item = arr[arr.Length - 1];
Array.Resize(ref arr, arr.Length - 1);
return item;
 }
```

The Makenull method is implemented through the built-in capability of the collection System.Array using a stack-size reduction loop until it is reduced to zero:

```
publicvoidMakenull()
 {
for (int i = arr.Length; i > 0; i--)
 {
Array.Resize(ref arr, arr.Length - 1);
}
 }
 }
```

This solution of adding and deleting elements leads to using the exact needed size of ADT.

III. RESULTS

The base class, showing a new method of creating Abstract Data Types, was implemented. The size of memory, used to store elements, was decreased. No compilation errors founded. All the tasks have been completed. The new method has shown the advantage in terms of hard drive memory saving.

3.1 Program Code (C#)

```
using System;
usingSystem.Collections.Generic;
using System.Linq;
using System.Text;
usingSystem.Threading.Tasks;

namespace MyStack
{
classMyStack<T>
 {
T[] arr;
public MyStack()
 {
 arr = newT[0];
 }
public bool Empty
 {
get
 {
return arr.Length> 0;
 }
 }
public voidPush(T item)
 {
Array.Resize(ref arr, arr.Length + 1);
arr[arr.Length - 1] = item;
 }
public T Pop()
 {
 T item = arr[arr.Length - 1];
Array.Resize(ref arr, arr.Length - 1);
return item;
 }
publicvoidMakenull()
 {
for (int i = arr.Length; i > 0; i--)
 {
Array.Resize(ref arr, arr.Length - 1);
 }
 }
 }
classProgram
 {
staticvoid Main(string[] args)
```

```
 {
MyStack<int> s1 = newMyStack<int>();
MyStack<int> s2 = newMyStack<int>();
for (int i = 0; i < 10; i++)
 {
 s1.Push(i);
Console.Write(i + " ");
}
Console.WriteLine();
for (int i = 10; i > 5; i--)
 {
int a = s1.Pop();
 s2.Push(a);
 }
while (s1.Empty)
Console.Write(s1.Pop() + " ");
Console.WriteLine();
while (s2.Empty)
 {
Console.Write(s2.Pop() + " ");
 }
Console.WriteLine();
s2.Makenull();
Console.WriteLine();
while (s2.Empty)
 {
Console.Write(s2.Pop() + " ");
 }
Console.WriteLine();
for (int i = 0; i < 10; i++)
 {
 s2.Push(i);
Console.Write(i + " ");
 }
Console.WriteLine();
 }
 }
}
```

IV. DISCUSSION

As a result, this method shown and proven itself as a good alternative to previous one. The size of memory, used to store elements, was decreased. As a negative side has been founded a small increase of RAM usage, but using multithreaded core can solve this problem. That way, previewed method is recommended for use on big projects with many elements in it, while Microsoft Visual Studio's solution is recommended for use on small number of elements. In that case, both methods proven their usability, but both of them also have negative sides.

V. CONCLUSION

As a result, this method shown and proven itself as a good alternative to previous one. The size of memory, used to store elements, was decreased. Accumulated experience confirmed the extensibility solution of the Abstract Data Types and encouraged further exploration within other application domains, such as saving information and tables file creation. The article have revealed a proprietary mechanism that was closely related with the Abstract Data Types approach for specifics of object-oriented type of programming. Therefore, the article considered the implementation of the new way of saving memory in the context of crating files using Abstract Data Types, especially for Microsoft Visual Studio soft.

ACKNOWLEDGEMENTS

I have been fortunate to work with Olga Popova, talented lecturer that inspired and influenced this research. Studies of this bright person let me knowledge about Abstract Data Types, terms of its usage and its structure. Lections and practical studies under her guidance and supervision had shown me the best way of using my own knowledge and abilities of searching for and discovering new aspects of programming. I am very thankful that I had luck to be her student.

REFERENCES

[1] Rod Stevens, (2016). Algorithms-theory and practical application (Linked list. Stacks and queues.). Moscow.: E publishing house, 544 p.

[2] Pavlovskaya T. A., (2009). C#. Programming in a high-level language. St. Petersburg: Peter, 432 p

[3] Microsoft MSDN open resource