Universal Linear Data Structure

Rustam Kabulov Vosilovich
Candidate of Physical and Mathematical Sciences, Associate Professor, department of software engineering
Tashkent University of Information Technologies named after Muhammad al-Khwarizimi, 100200, Republic of Uzbekistan, Tashkent, Amir Temur str., 108
kabulov_rustam1959@mail.ru

***Abstract*— The paper is devoted to the relevant issue of formalizing the description of a universal linear data structure. In modern programming, various data structures are described as linear data structures. The paper presents general functions for performing operations on such data structures as vector, deck, list, set and multiset. The conditions that these functions must meet are introduced as axioms. To describe the axioms, an approach based on an abstract data type, algebraic and formal grammatical approaches were used.**

Keywords— data structure, container, type, function, axiom, precondition, constructor, operation, associative container, iterator.

I. Introduction

The paper is devoted to the issue of formalizing the description of linear data structures presented in the form of container classes. Classes are called containers designed for storing elements of arbitrary type. Every modern object-oriented programming language contains a container class library. The best-known one is the C ++ STL library. The library includes, in addition to containers, iterators and memory allocators.

All containers in the STL are categorized into derivative, sequential, and associative containers. Derivative containers include stack, queue, and priority queue. Sequential ones include list, vector, and decks. Associative containers include a set, a multiset, a map, and a multicard. It should be noted that associative containers are represented as sequential containers using iterators[1-3].

This article uses several different approaches to describe universal integrated operations on these data structures[4-6]. The first approach uses the functional abstraction mechanism [7-10] in a modified form, close to real programming practice.

The article introduces functions for performing operations on sequential data structures, as well as associative arrays. The formalization of the concept of iterators as a mechanism of working with containers was considered. To describe axioms, in addition to the approach based on the abstract data type, algebraic and formal grammatical approaches were used. The algebraic approach uses set-theoretic operations and an element extraction operation[11-15]. The operation of counting the number of given elements was also used. In the formal grammatical approach, inference rules were introduced instead of axioms.

The work is theoretical and directed to formalize the description of sequential data structures and generalized specification description. Purpose of work: development of methods for formalizing description and specifications of sequential data structures as close as possible to real programming practice.

II. Functional Method

An abstract data type is described using types, functions, axioms, and preconditions. Mathematical notation is used to describe functions. Function parameters are universal (generic) data types. The functions do not change the values of the parameters and return the new value of the container.

Functions:
push_front: CONTAINER [T] $\times$T → CONTAINER [T]
push_back: CONTAINER [T] $\times$T → CONTAINER [T]
pop_front: CONTAINER [T] → CONTAINER [T]
pop_back: CONTAINER [AT] → CONTAINER [T]
front: CONTAINER [T] → T
back: CONTAINER [T] → T
empty: CONTAINER [T] → BOOLEAN
size: CONTAINER [T] → INTEGER
count: CONTAINER [T] $\times$ T → INTEGER
Default constructor and copy constructor.
CONTAINER: CONTAINER [T]
CONTAINER: CONTAINER [T] → CONTAINER [T]

The delete and read functions are not defined for an empty instance. This fact [4] is designated as:
pop_front (Q: CONTAINER [T]) require not empty (Q)
pop_back (Q: CONTAINER [T]) require not empty (Q)
front (Q: CONTAINER [T]) require not empty (Q)
back (Q: CONTAINER [T]) require not empty (Q)

General axioms:
For all x: T, Q: CONTAINER [T],
(A1) empty (CONTAINER())
(A2) not empty (push_front (Q, x))
(A3) not empty (push_back (Q, x))
(A4) size (CONTAINER())=0
(A5) CONTAINER(Q))=Q
(A6) size (push_front (Q, x))=size(Q)+1
(A7) size (push_back (Q, x))=size(Q)+1
(A8) size (pop_front (Q))=size(Q)÷1
(A9) size (pop_back (Q))=size(Q)÷1
If a>b then a÷b=a-b, otherwise 0

Counting axioms:
For all x: T, Q: CONTAINER [T],
(A1) count (CONTAINER(),x)=0
(A1) count (push_front (Q, x),x)=count(Q,x)+1
(A2) count (push_back (Q, x),x)=count(Q,x)+1
(A3) count(pop_front (Q),front(Q))=count(Q,front(Q))÷1
(A4) count (pop_back (Q),back Q))=count(Q,back(Q))÷1

Axioms:
For all x: T, Q: CONTAINER [T],
(A1) front (push_front (Q, x)) = x
(A2) back (push_back (Q, x)) = x
(A3) pop_front (push _front(Q, x)) = Q
(A4) pop_back (push _back (Q, x)) = Q

By P (Q) we denote the set of elements in the container.
If P(Q)={a0,…,an} then:
front (Q)=a0
back(Q)=an

P(pop_front(Q))= {a1,…,an}
P(pop_back (Q))= {a0,…,an-1}
P(push_front(Q,b))= {a0,…,an,b}
P(push_back (Q,b))= {b,a0,…,an}.

III. ALGEBRAIC DESCRIPTION

In the algebraic approach, together with functions, algebraic operations on abstract data types or containers are considered.

Operations:
A + B addition
A-B delete
A ÷ B delete
A * read an item
* A read an item
Axioms:
For all A, B: CONTAINER [T],
(A1) (A+B)* = B*
(A1) *(A+B) = A*
(A2) (A+B)-B = A
(A2) (A+B) ÷A = B
If P(A)={a0,…,an} then:
*A=a0
A*=an
If P(A)={a0,…,an} and P(B)={b0,…,bn} then:
A+B={a0,…,an, b0,…,bn}
If P(C) ={a0,…,an, b0,…,bn} and P(A)={a0,…,an} and P(B)={b0,…,bn} then:
C - B={a0,…,an }
C÷A={b0,…,bn}.

IV. ITERATORS

Iterators are the closest approach to real programming practice. Iterators are usually divided into active and passive ones. Active ones can change the container they are associated with, passive ones cannot. Consider the axioms for a one-way passive iterator.

Functions for a one-way passive iterator:
next: ITERATOR [T] → ITERATOR [T]
haQ: ITERATOR [T] → BOOLEAN
value: ITERATOR [T] → T
Axioms for iterators:
For all A, B: ITERATOR [T],
(A1) A=B → next(A)=next(B)
(A2) next(A)=next(B) →A=B
(A3) A=B → value(A)=value(B)
(A4) not haQ (A) →next(A)=A
Axioms connecting container and iterator:
For all A: ITERATOR[T], Q: CONTAINER [T],
A=iterator(Q) → count(Q,*A)>0
count(Q,*A)>0→ A=iterator(Q).

V. ALGEBRAIC APPROACH

Unary operations:
- ++A forward step
- --A backward step
- *A value

Axioms for iterators:
For all A, B: ITERATOR [T],
- (A1) A=B → ++ A=++ B
- (A1) ++A=++ B →A=B
- (A1) A=B → -- A= -- B
- (A1) --A= -- B →A=B
- (A1) A=B → *A=*B
- (A1) -- ++A= A
- (A1) ++ --A=A

VI. INDICES

Axioms connecting container and index:

For all P,Q: ITERATOR[T], i: INTETER,
- P=Q → ∀(i<size(P)) (P[i]=Q[i])
- ∀(i<size(P))(P[i]=Q[i]) → P=Q
- i<size(A)→count(P,P[i])>0
- count(P,y) →∃i(y=P[i])

Axioms connecting associative container and index:
For all P,Q: ITERATOR[T], I: F,
- P=Q → ∀(I) (P[I]=Q[I])
- ∀(I)(P[I]=Q[I]) → P=Q
- ∀(I) (count(P,P[I])>0)
- count(P,y) →∃I(y=P[I])

VII. FORMAL GRAMMATICAL METHOD

In the formal grammatical approach, inference rules are introduced instead of axioms. Terminals correspond to the items stored in the container.

Grammar rules for the container are:
- (A1) Q → AB
- (A1) A →Aa
- (A1) B → aB
- (A1) Aa →A
- (A1) aB→B

Grammar rules for a two-way active iterator:
- (A1) Qa → aQ
- (A1) aQ →Qa
- (A1) Qa → Q
- (A1) Q →aQ

VIII. CONCLUSION

The article introduces general functions for linear data structures. The introduced evaluation functions make it possible to put aside excessive detail and more fully take into account the requirements for specific implementations. The considered formalization methods are close to real practice.

Results of work:

The mechanism of functional abstraction has been developed in a modified form, close to real programming practice.

An algebraic approach has been developed that can be implemented in programming practice based on the operation overloading mechanism.

A formal grammatical apparatus has been developed using grammatical inference rules to represent the specification of functions of data structures.

In the future, it is planned to use the developed approaches to describing nonlinear data structures used in real programming.

REFERENCES

[1] Alexandrescu A. Modern C++ design. The C++ InDepth series, vol. 3 / TRANS. from English. — M.: William, 2002. — 336 S., ill.

[2] Ammeraal L. STL for C++ programmers / TRANS. from English. — M.:DMK, 1999. - 240 p., Il.

[3] Ostern M. Generic programming and the STL: The use and capacity of the standard template library / Per. with angl.- SPb.: Nevsky dialect, 2004.- 544 p., Il.

[4] Aho A. V, Hopcroft J. E., Ullman J. D. Data structures and algorithms = Data Structures and Algorithms. Williams, 2000. — 384 C.

[5] A. B. Tucker and R. E. Noonan. Programming Languages: Principles and Paradigms, Second Edition. McGraw-Hill Higher Education, 2007.

[6] Knut D. The art of programming. Volume 1. Basic algorithms. Williams, 2010.

[7] Liskov B., Zilles S. Programming with abstract data types // SIGPlan Notices, vol. 9, no. 4, 1974.

[8] B. Liskov. A history of CLU. In History of programming languages—II, pages 471–510. ACM, 1996

[9] L. Cardelli. A semantics of multiple inheritance. In Semantics of Data Types, volume 173 of Lecture Notes in Computer Science, pages 51–68. Springer-Verlag, 1984.

[10] Bertrand Meyer. The basics of object-oriented programming. Abstract data types (ADT). http://www.intuit.ru/department / se / oopbases / 6 / 10.html .

[11] B. C. Pierce. Types and Programming Languages. MIT Press, 2002.

[12] Glushkov V. M., Zeitlin, G. E., Yushchenko E. L. Algebra. Languages. Programming. 3-e Izd., Rev. and DOP. – Kiev: Sciences. Dumka, 1989. -376 p.

[13] Yu. V. Kapitonova, A. A. Letichevskii, and V. A. Volkov, "Deductive tools of an algebraic programming system," Cybernetics and System AnalysisNo. 1, 12-26 (2000).

[14] Kryvyi S. L. Abstract data types as polybasic algebraic system. – K.: well. "Software engineering". – 2010. - № 3. – C. 3 -18.

[15] A. E. Doroshenko, O. V. Iovchev A method of designing an abstract data type in algebra of Algorithmics // Problems of programming. – 2012. – No. 1. – P. 3 – 16.

[16] Kabulov R.V., Latipova N. Axiomatic approach to the definition of abstract types of data. // "Important issues in the field of technical and socio-economic sciences." Republican collection of interuniversity scientific works. Tashkent Institute of Chemistry and Technology. Tashkent. 2015 pp.48-49.