

## АБСТРАКТНИ ТИПОВЕ ДАННИ (СТЕК, ОПАШКА) – МЕТОДИЧЕСКИ АСПЕКТИ В ПРЕПОДАВАНЕТО ИМ

**Ирина Христова**

Минно-геоложки университет „Св.Иван Рилски“, 1700, София, [irinahr@mgu.bg](mailto:irinahr@mgu.bg)

**РЕЗЮМЕ:** Настоящият доклад е посветен на методическите аспекти в обучението по „Абстрактни типове данни – стек и опашка“. Препоръчаният метод на обучение според авторката води до повишаване на неговата ефективност.

### ABSTRACT DATA TYPE (STACK, QUEUE) – METHODOLOGICAL ASPECTS OF IT TEACHING

**Irina Hristova**

University of Mining and Geology "St. Ivan Rilski", 1700 Sofia, [irinahr@mgu.bg](mailto:irinahr@mgu.bg)

**ABSTRACT.** The present report is devoted to the methodological aspects of teaching the subject "Abstract data – stack and queue". The recommended method of teaching in the view of the author lead boosting it effectiveness.

### Въведение

Абстрактните типове данни имат голямо значение за съвременното програмиране. Използват се за описание на различни системи, явления, предмети. Абстракцията се основана на разделянето на методите на използване на данните от методите на тяхното конкретно представяне. Принципът на определяне на типа данни е чрез операциите, които могат да се изпълняват над обектите от даден тип, като се въвежда ограничението: стойностите на тези обекти могат да се изменят и извличат само по пътя на използването на тези операции.

Стекът и опашката са типични представители на абстрактните типове данни (АТД) с линейна наредба. За тях съществуват както хардуерна, така и програмна реализация. Последната може да бъде:

- статична – представяне чрез основните типове данни, поддържани в езиците за програмиране – масив;
- динамична – чрез указатели.

Усвояването им в известна степен затруднява обучаваните, поради определеното ниво на абстракция.

В настоящият доклад се описва подход, при който в обучението се използват готови програми за демонстриране на основните операции над стек и опашка. Сорс файлът е написан на езика за програмиране C++, като:

**Stack** - програма, демонстрираща стек и основните операции със стек;

**Queue** - програма, демонстрираща опашка и основните операции със опашка.

Описанието на абстрактните типове данни е осъществено чрез динамични рекурсивни структури.

Диалогът с обучаемите е реализиран чрез подходящо меню. На фиг.1 е представен част от диалога с програмата Stack.

```
C_reate, P_ush, pO_p, T_op, pR_int, E_mpty, exaM_ple, eX_it p
Enter item.inf: 12
C_reate, P_ush, pO_p, T_op, pR_int, E_mpty, exaM_ple, eX_it p
Enter item.inf: 13
C_reate, P_ush, pO_p, T_op, pR_int, E_mpty, exaM_ple, eX_it r
Print a stack with destroy (Y/N), Q_uit: n
112
Print a stack with destroy (Y/N), Q_uit: q
Quit printing the stack.
C_reate, P_ush, pO_p, T_op, pR_int, E_mpty, exaM_ple, eX_it p
Enter item.inf: 14
C_reate, P_ush, pO_p, T_op, pR_int, E_mpty, exaM_ple, eX_it o
14
C_reate, P_ush, pO_p, T_op, pR_int, E_mpty, exaM_ple, eX_it x
Exit program.
Press any key to continue
```

фиг. 1

На студентите се представят сорс файла на програмите, като обръща се внимание на двете фази при проектирането на абстрактните типове данни:

описание (представяне) на данните – със средствата на съответния език за програмиране.

описание на операциите, допустими за съответния тип данни.

Основният акцент е разбиране и усвояване на основните операции: включване и изключване на елемент, съответно за стек и опашка. За целта подходящи схеми при тяхното преподаване.

Знанията за тези типове данни се затвърдяват чрез допълнителни задачи, чиято програмна реализация се осъществява от обучаемите.

### СТЕК

Обучението протича в следната последователност:

#### 1. Дефиниране на абстрактния тип данни стек:

Стекът е крайно линейно множество от еднотипни елементи с дисциплина LIFO („Last In – First Out” – „последен включен - пръв изключен”). Операциите включване и изключване на елемент се извършва само в единият край на множеството, наречен връх.

#### 2. Представяне на абстрактния тип данни стек.

Елементите на стека могат да бъдат произволни, но обикновено са от един тип с фиксирана дължина. Организацията е последователна, като директен достъп е допустим само до елемента, намиращ се на върха на стека. Управлението се осъществява чрез указател, сочещ върха на стека.

Съществуват два начина на за физическо представяне на стека:

2.1. Последователно представяне – заделя се блок от паметта, в който се съхраняват елементите на стека. В езиките за програмиране това се осъществява чрез вектор (структура от данни – масив, която може да се използва за представяне на стека), показано на фиг.2. Броят на елементите е строго фиксиран.

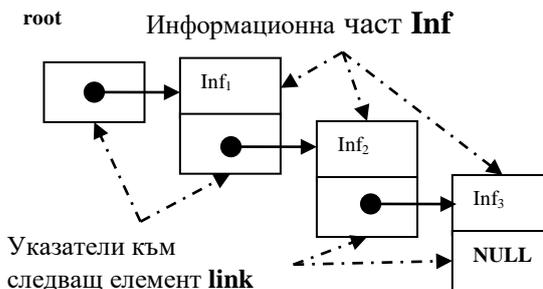


фиг. 2

2.2. Свързано представяне (фиг.3) – Стекът се представя като линейна структура от елементи. Всеки елемент се състои минимум от две полета:

- Информационна поле (**Inf**)– носител на информацията, която трябва да се съхранява;
- Указател към следващ елемент (**link**).

Предварително не се заделя памет, а тя се отделя динамично по време на изпълнението на програмата, когато елементите на стека започнат да съществуват. Броят на елементите динамично се променя.



фиг. 3

#### 3. Описание на стек със средствата на ЕП С++.

За представянето на един елемент от стека се използва структурата **Item**, показана фиг.4. Нейните полета са съответно:

- **Inf** – поле, носител на информацията, която ще се натрупва в стека. Може да бъде от представя произволен тип данни.
- **link** – указател към следващ елемент.

Структурата е рекурсивна. В дефиницията си съдържа обръщение към нейното име - **Item\* link**.

```
struct Item
{int Inf;
Item* link;
} *root, *p;
```

фиг. 4

След описанието на структурата са деклариран две променливи:

- \***root** – указател към върха на стека;
- \***p** – работна променлива.

#### 4. Операции със стек:

##### 4.1. Включване на елемент в стек.

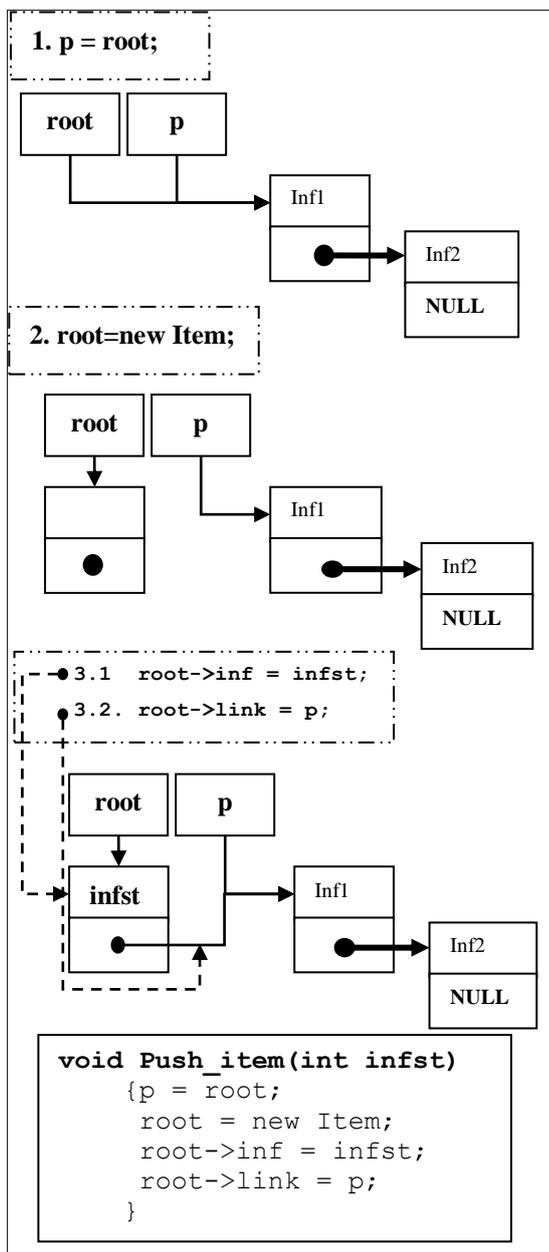
Включването на елементи в стека се осъществява в реда, в който постъпват.

Схемата и функцията, реализиращи операцията са показани на фиг. 5. Функцията има параметър-стойност **infst**, чрез който се предава информационната част на включвания елемент.

##### 4.2. Изключване на елемент от стек.

Изключването на елементи в стека се осъществява в ред, обратен на тяхното постъпване.

Схемата и функцията, чрез които се осъществява операцията изключване на елемент са дадени на фиг. 6. Функцията има параметър **&infst**, който връща информационната част на изключвания елемент и помощна локална променлива от тип указател **\*q**, необходима за реализиране на операцията. Във функцията се извършва проверка дали стека е празен и ако това е така извежда подходящо съобщение: "The stack is empty!".



фиг. 5

#### 4.3. Допълнителни операции:

Програмата **Stack** включва функции, които демонстрират действието на стека.

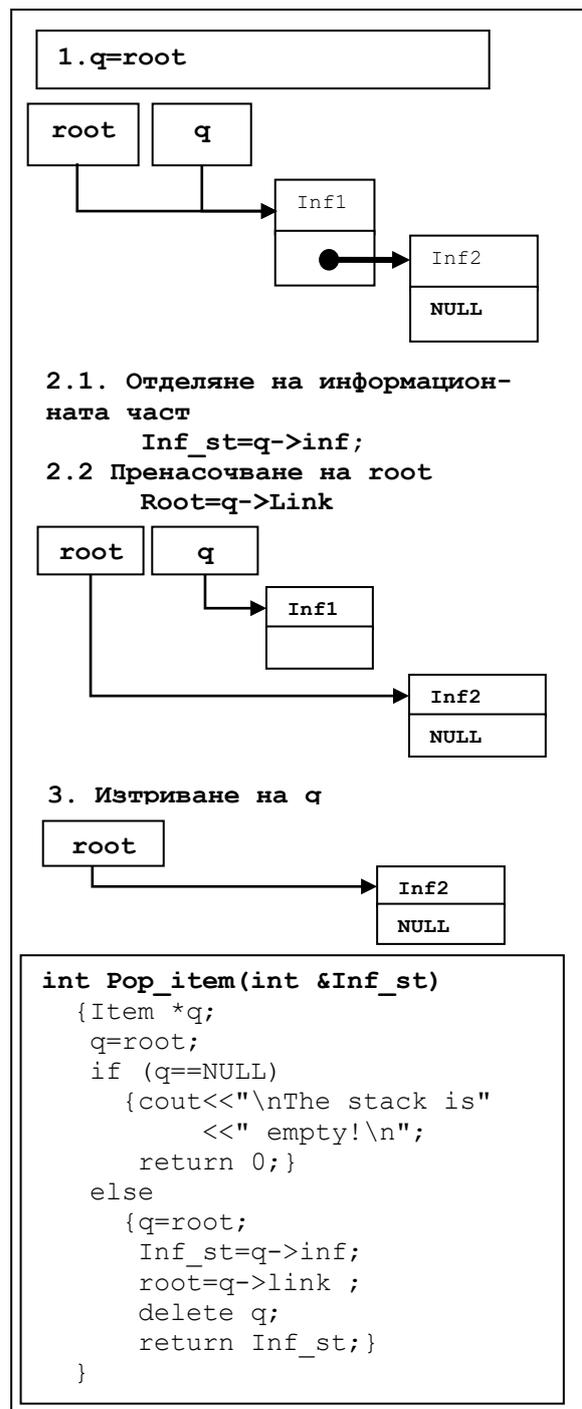
□ Отпечатване на елементите, включени в стека – предвидени са две функции **PrintStack()**, отпечата елементите на стека без да го разрушава; и функцията **PrintDestroyStack()** - отпечата елементите на стека, като го разрушава.

□ Извличане на елемента, намиращ се на върха на стека - **Top\_item(int &inf\_st)**:

- Изтриване на елементите на стека - **DeleteStack()**;
- Създаване на празен стек: **-Create\_empStack()**;
- Проверка за празен стек - **Empty\_stack()**.

#### 5. Затвърдяване на знанията.

Като допълнителна задача се задава: Преобразуване на цели десетични числа в двоични.



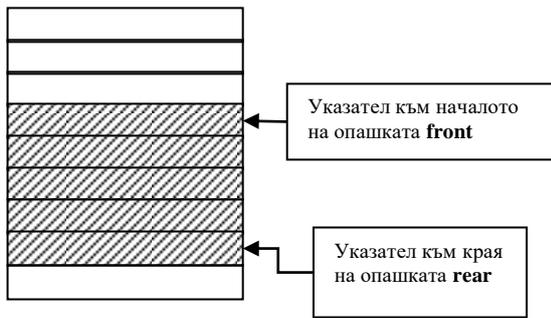
фиг. 6

#### ОПАШКА

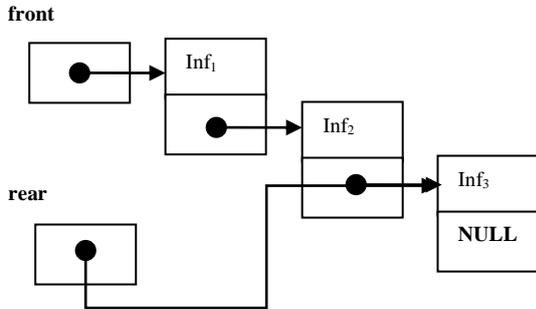
Описаният подход се прилага и при преподаването на опашка, като се прави сравнителен анализ между двата типа структури данни стек и опашка.

#### 1. Дефиниране и представяне на опашка

Опашката подобно на стека е линейна структура, но с дисциплина FIFO (First In – First Out – „първ включен - първ изключен“). Операциите включване и изключване на елемент се извършват в двата различни края на опашката. Статичното представяне на опашката е показано на фиг. 7, а на фиг. 8 – динамичното представяне.



фиг. 7



фиг. 8

## 2. Описание на опашка със средствата на езика C++

Структурата е аналогична на тази на стека. След декларацията на опашката са дефинирани две променливи:

\*front – указател към началото на опашката;

\*rear – указател към края на опашката.

```
struct Item
{int inf;
 Item* link;}
*front, *rear;
```

фиг. 9

## 3. Операции с опашка

### 3.1. Включване на елемент в опашката.

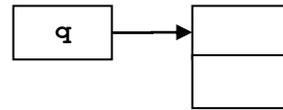
Включването на елемент в опашката се осъществява в нейния край. Схемата на включване и функцията, която я реализира е показана на фиг.10

### 3.2. Изключване на елемент от опашката

Изключването на елемент от опашката се осъществява от нейното начало. Механизмът е аналогичен на този на изключване на елемент от стек, затова тук се дава само функцията, която реализира тази операция:

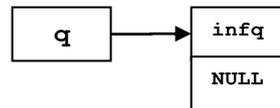
```
int Pop_from_queue()
{Item *q;
 int itq_inf;
 q=front;
 itq_inf=q->inf;
 front=front->link;
 return itq_inf;
 delete q;
}
```

### 1. Създаване на нов елемент q=new Item



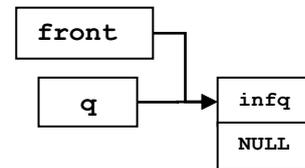
### 2. Установяване на стойностите на този елемент:

```
q->inf = infq;
q->link = NULL
```

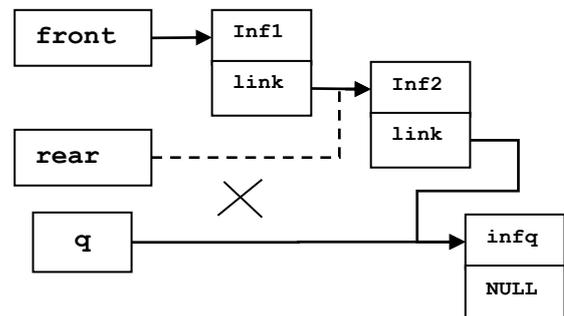


### 3. Включване на елемента

а) опашката е празна  
front = q;



б) опашката не е празна  
rear->link = q;



```
void Push_in_queue(int infq)
{Item *q;
 q=new Item;
 q->inf=infq;
 q->link =NULL;
 if (front==NULL)
     front=q;
 else
     rear->link = q;
 rear =q;
}
```

фиг. 10

### 3.3. Допълнителни операции

□ Отпечатва съдържанието на опашката без да я разрушава - **Print\_queue()**;

□ Извличане на елемент, находящ се в началото на опашката - **Front\_queue()**;

□ Извличане на елемент, който в края на опашката - **Rear\_queue()**;

- Определяне размера на опашката - **Size\_of\_queue();**
- Изтриване на опашка – изтриват се всички нейни редове - **DeleteQueue();**
- Създаване на празна опашка - **Create\_empty\_queue();**
- Проверка за празна опашка - **Empty\_queue();**

#### 4. Затвърдяване на знанията

Като допълнителна задача за затвърдяване на знанията се задава: Определяне на поредния номер (индекса) на предварително зададен елемент от опашката.

## Заклучение

Описаният подход на преподаване дава възможност да изследват основните операции с двата типа структури от данни стек и опашка и да се избегнат затрудненията при тяхното овладяване от обучаемите.

## Литература

Тодорова, М. 2004. *Програмиране на C++*. Част Втора. С., Ciela. 483 с.  
Уирт, Н. 1980. *Алгоритми + структури от данни = програми*. С., Техника, 390 с.

Препоръчана за публикуване от катедра "Информатика"  
МЕМФ