

КАТЕДРА: КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ
ДИСЦИПЛИНА: АЛГОРИТМИ СТРУКТУРИ ОТ ДАННИ
ДИСЦИПЛИНА: СИНТЕЗ И АНАЛИЗ НА АЛГОРИТМИ

ЛАБОРАТОРНО УПРАЖНЕНИЕ № 3

ТЕМА: Линејни структури от данни

ЦЕЛ:

Целта на упражнението е студентите да се запознаят със линейните структури от данни. След упражнението студентите би следвало да могат да боравят със списъци да създават стекове и опашки.

I. ТЕОРЕТИЧНА ЧАСТ

1. Въведение.

В програмирането, както и в математиката, доста често се налага да се обработват дадено множество от елементи. Пример за това е прибавяне на елемент към дадено множество, премахване на елемент от множество, подреждане на множество по даден закон или даден показател. Структурите от данни са една основна концепция за съхранението и организацията на данните в компютърната наука.

Структурите от данни могат да бъдат разглеждани като математически, логически модел, по който са организирани данните или абстрактни типове данни. Детайлното познаване на тази концепция води до създаване на по-ефективен и по-структуриран програмен код. Един добре познат пример за структура от данни е масивът. Както добре знаем, масивите са линейни структури от данни, записани последователно в паметта, като всеки един елемент притежава индекс и достъпа до даден елемент става чрез неговия индекс. Основната идея на упражнението е да се разгледат някои от останалите линейни структури от данни-списъци, стекове и опашки.

Списъкът е линейна структура от данни, която съдържаща в себе си поредица от елементи. Различава се от масива по това, че може да се оразмерява динамично. Списъците имат свойството дължина (брой елементи) и елементите му са наредени последователно. С помощта на имплементираните към него методи, е възможно добавянето на нови елементи на която и да е позиция в списъка, махането на такива, обхождането или обръщането на тези елементи и т.н.

Всички елементи в списъка са свързани по някакъв начин, като всеки елемент има указател който сочи към следващия елемент от списъка и ако е необходимо и към предишния. В зависимост от това списъкът може да бъде едно-свързан и двусвързан. Едно-свързания списък е този чиито елементи имат само един указател, който сочи следващия елемент, а двусвързания – елементите му имат два указателя като единия сочи предходния а другия следващия елемент. Дефинирането на списък става с обикновена структура от данни. В долния пример е показано дефинирането на едносвързан списък, като дефинирането на двусвързания е по подобен начин.

struct elem

```
{    int key; // поле в което съхранява стойността на съответния елемент от списъка  
    elem *next; //указател към следващия елемент от списъка
```

```
} *start;    // указател сочи първия елемент от структурата
```

Списъците имат следните операции към тях.

- Add(items) – Добавяне на елемент към списъка;
- Del(items) - Изтриване на елемент от списъка;
- Search() – Търсене на елемент от структурата;

Първите две операции имат няколко зависимости в зависимост от това къде се добавя или изтрива елемента. Дали в началото в края или някъде по средата. Тези операции бяха разгледани подробно на лекциите. В Практическата част и задачите за самостоятелна работа е необходимо да се използват някои от тези функции.

Стекът представлява структура от данни с поведение "последен влязъл първи излизал". За неговото обяснение може да си представим една ракла в която слагаме завивки и върху тях дрехи. За да се извадят завивките първо трябва да се извадят всички дрехи върху тях.

Операциите със стек са следните:

- Добавяне на елемент в стека;
- Изваждане на елемент в стека;

В стекът първо се вади елементът който е на върха на стека, след което следващия и след него и т.н. Не е възможно да се извади друг елемент освен този който е на върха. Добавянето на нов елемент в стека задължително се поставя на върха на стека. Дефиницията за структурата стек е същата като едносвързан списък с тази разлика че полето за указател в структурата тук указва следващия елемент който следва да се извади от стекът.

Опашката представлява структура от данни с поведение "първият влязъл първи излизал". Тази структура реализира опашка, като например чакащи документи за принтиране, чакащи процеси за достъп до общ ресурс и други. В опашката може да се добавя елемент само най – отзад и да се извлича елемент само най – отпред. Най основните операции за работа с опашки са следните:

- Добавяне на елемент в края на опашката;
- Изваждане на елемент от началото на опашката

Структурата за опашка е подобна като тази за стек и списък с тази разлика че трябва да се дефинират два указателя от тази структура сочещи първия и последния елемент на опашката.

II. ПРАКТИЧЕСКА ЧАСТ

В практическата част са показани примери, които показват как се използват линейните структури от данни.

ЗАДАЧА1: Да се напише алгоритъм за намиране на простите числа в определен интервал.

КОД:

```
.....  
void Getprime(int start, int end)  
{  
    for (int num = start; num <= end; num++)  
    {  
        bool prime = true;
```

```

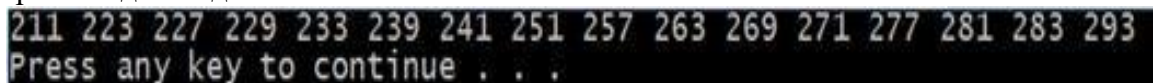
        double numSqrt = sqrt(num);
        for (int div = 2; div <= numSqrt; div++)
        {
            if (num % div == 0)
            {
                prime = false;
                break;
            }
        }
        if (prime)
        {
            add(num); // функция добавяща елемент в края на списъка
        }
    }
}
....
static void Main()
{
    Getprime(200, 300);
    print(); /* функция разпечатваща първите елементи от масива и освобождава паметта
от съответния разпечатан елемент*/
}

```

ПОЯСНЕНИЕ

От математиката е известно че ако едно число не е просто то съществува поне един делител в интервала от 2 до корен квадратен на самото число. Този алгоритъм се реализира чрез горния код. За всяко изследвано число се търси дали съществува делител в този интервал. Ако се срещне делител то числото не е просто и се продължава към следващото число. Ако числото е просто, то се добавя към списъка.

След като са добавени всички прости числа от изследвания интервал, то чрез функция print() се разпечатват първото число от списъка и се освобождава паметта от него. Това продължава докато има елементи в списъка. Конкретния пример в задачата търси всички прости числа в интервала между 200 и 300. Резултат от изпълнението на горния код е следния:



```

211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
Press any key to continue . . .

```

Фиг.1. Резултат от изпълнението на кода за зад.1.

ЗАДАЧА2: Да се създаде стек. Да се добавят няколко елемента в него след което да се изведат.

КОД:

```

Struct elem {
String key;
Elem *next;
} *start=NULL;
.....
public static void Main()
{
    push("1. Ivan");
    push("2. Nikolay");
    push("3. Maria");
    push("4. George");
}

```

```

string n;
do {
    n=pop();
    cout << n << endl;
} while (n != "empty");

```

ПОЯСНЕНИЕ

Тъй като стекът е структура „последен влязъл – пръв излязъл“, то програмата ще изведе записите в ред обратен на реда на добавянето. В стека се добавят следните имена “Ivan, Nikolay, Maria, George”. За улеснение към имената е добавен номер под формата на низ, който съответства на реда в който постъпват в стека. Функцията push добавя елемент в стека, а функцията pop извлича елемента от стека, като освобождава заделената памет за него. Резултатът от програмата ще бъде следния:

```

Top = 4. George
4. George
3. Maria
2. Nikolay
1. Ivan
Press any key to continue . . .

```

Фиг.2. Резултат от изпълнението на кода от зад. 2.

ЗАДАЧА3: Да се създаде опашка. Добавете в нея няколко елемента. След което изведете всички чакащи елементи в конзолата.

КОД

```

public static void Main()
{
    Add_que("Message One");
    Add_que("Message Two");
    Add_que("Message Three");
    Add_que("Message Four");

    while (queue.Count > 0)
    {
        string msg = Remove();
        cout << msg << endl;    }
}

```

ПОЯСНЕНИЕ

Резултатът от изпълнението на горния код е следния:

```

Message One
Message Two
Message Three
Message Four

```

Фиг. 3. Резултат от изпълнението на кода за зад. 3.

Вижда се че елементите излизат от опашката в реда, в който са постъпили в нея.

III. Задача за самостоятелна работа.

1. Тествайте задача1 от практическата част. За целта на място на многоточието е необходимо да реализирате структура за едносвързан списък, както и да дефинирате функциите за добавяне на елемент към края на списъка, и функция print(). Тази функция трябва да изтрива елемента от динамичната памет като преди това го разпечатва.
2. Тествайте задачи 2 и 3 от практическата част. За целта на мястото на многоточието е необходимо да реализирате структурите за стек/опашка, както и функциите за добавяне и премахване на елемент от тях.
3. Като използвате структурата за списъци създайте програма която може да извършва обединение и сечение на две множества от числа.

Упътване: Множествата ги представете като списъци.

4. Като използвате структурата за стек, напишете програма, която да проверя дали броя на отварящите скоби съответства на броя на затварящите скоби за следния числов израз:

`"1 + (3 + 2 - (2+3) * 4 - ((3+1)*(4-2)))";`

Упътване: Използвайте Стек. Когато срещнете отваряща скоба я добавяте в стека, а когато срещнете затваряща скоба вадите елемент от стека. Ако стекът остане празен преди алгоритъма да е приключил, в момент в който трябва вадите още един елемент, значи скобите са некоректно поставени. Същото важи и ако в края в стека останат някакви елементи.