

Algoritmi i strukture podataka 2019./2020.

Zadatci za vježbu – liste

1. Napišite strukturu podataka koja realizira stog proizvoljnog tipa `T` poljem čija se veličina može dinamički mijenjati.

- a) Napišite dva konstruktora u kojima se rezervira inicijalna količine memorije i inicijalizira stog. Prvi konstruktor treba biti podrazumijevani konstruktor i rezervirati količinu memorije za pohranu 100 vrijednosti tipa `T`, dok drugi treba od korisnika primiti veličinu `size` te rezervirati količinu memorije za pohranu `size` elemenata na stog.
- b) Napišite funkcije za dodavanje i skidanje elementa sa stoga koje dinamički povećavaju i smanjuju korištenu količinu memorije kako se stog povećava ili smanjuje. Količina memorije se povećava ili smanjuje za `increment = 100`.

2. Koristeći stog implementiran zadatkom 1), napišite funkciju koja vraća prvi član na stogu, ali tako da nakon izlaska iz funkcije taj član ostane na stogu (funkciju treba nazvati `peek`). Smiju se koristiti samo osnovne funkcije za rad sa stogom (`dodaj` i `skini`) koje su napisane u prethodnom zadatku (nije dozvoljen pristup privatnim članovima klase `Stack<T>`).

3. Napišite novu funkciju u klasi koja implementira stog listom (dinamička alokacije memorije). Funkcija vraća `i`-ti član na stogu (`i = 0` označava vrh stoga). Stog nakon izlaska iz funkcije mora ostati nepromijenjen, a smiju se koristiti samo osnovne funkcije za rad sa stogom `dodaj` i `skini`. Funkcija treba imati prototip:

```
bool peek(int &el, int index);
```

4. Napišite funkciju koja vraća prvi član u redu, ali tako da nakon izlaska iz funkcije taj član ostane u redu (funkciju treba nazvati `peek`). Smiju se koristiti samo osnovne funkcije za rad sa redom (`dodaj` i `skini`) (nije dozvoljen pristup privatnim članovima klase `Queue<T>`).

5. Napišite novu funkciju u klasi koja implementira red listom (dinamička alokacije memorije). Funkcija vraća `i`-ti član u redu (`i = 0` označava izlazni član u redu). Red nakon izlaska iz funkcije mora ostati nepromijenjen, a smiju se koristiti samo osnovne funkcije za rad s redom `dodaj` i `skini`. Funkcija treba imati prototip:

```
bool peek(int &el, int index);
```

6. (Zadatak sličan zadatku s **međuispita ak god. 2017/2018**) Zadani su tipovi podatka `Stack<T>` i `Queue<T>` za koji su definirani konstruktor, dodavanje elementa te skidanje elementa. Prototipovi navedenih funkcija su:

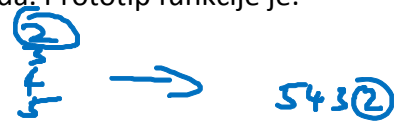
```
template <class T> class Stack {  
    public:  
        Stack();  
        bool push(T item);  
        bool pop(T &item);  
};
```

```
template <class T> class Queue {
    public:
        Queue();
        bool enqueue(T data);
        bool dequeue(T &data);
};
```

Funkcije `push` i `enqueue` vraćaju `true`, ako je operacija uspjela (bilo je dovoljno raspoložive memorije za dodavanje u stog ili red), a `false` ako nije. Funkcije `pop` i `dequeue` vraćaju `true`, ako je element uspješno skinut sa stoga ili iz reda, a `false` ako su stog ili red bili prazni (skidanje nije bilo moguće).

Napišite **rekurzivnu** funkciju koja će sve elemente iz stoga zadanog ulaznim parametrom prebaciti u red na način da element koji je bio na vrhu stoga bude prvi na izlazu iz reda. Prototip funkcije je:

```
template <class T>
bool transfer(Stack<T> *s, Queue<T>* q);
```



Ulazni stog nakon izvršavanja funkcije `transfer` ostaje prazan. Funkcija `transfer` vraća `true`, ako su svi elementi uspješno prepisani iz stoga u red, a `false` ako nisu (to se događa ako je prilikom dodavanja barem jednog elementa u red došlo do greške). Ako prilikom prepisivanja elemenata u bilo kojem trenu dođe do greške, potrebno je obustaviti daljnje rekurzivne pozive i povratkom iz rekurzije u glavni program vratiti `0`.

Napišite i dio glavnog programa u kojem ćete definirati po jedan red i stog, napuniti stog brojevima od `0` do `10` i ispravno pozvati napisanu funkciju.

VAŽNO: Nerekurzivno rješenje funkcije donosi najviše 1 bod.

7. (Zadatak sličan zadatku s **međuispita ak god. 2016/2017**) Zadan je tip podatka `Queue` za koji su definirane funkcije za inicijalizaciju reda, dodavanje elementa u red te za skidanje elementa iz reda. Prototipovi navedenih funkcija su navedeni u prethodnom zadatku.

Napišite funkciju koja će sva pojavljivanja višekratnika zadanog broja iz zadanog reda izdvojiti u novi red koji treba vratiti pozivatelju. Prototip funkcije je:

```
template <class T>
Queue<T> *selectMultiples(Queue<T> *q, int number);
```

U ulaznom redu moraju ostati svi preostali elementi.

Napišite i dio glavnog programa u kojem ćete definirati redove i ispravno pozvati napisanu funkciju. Dio koda u kojem se puni red nije potrebno pisati. Primjer: Za ulazni red sadržaja `3, 8, 2, 5, 35, 2, 24, 15` poziv funkcije `izdvojiVisekratnike` za broj=`3` stvara novi red sadržaja `3, 24, 15` dok u starom redu ostaju `8, 2, 5, 35, 2`.

Rješenja

```
1.
template <class T> class Stack {
private:
    static const size_t increment = 100;
    size_t size = 100;
    T *stack;
    int top = -1;

public:
    Stack() { stack = (T *)malloc(size * sizeof(T)); }
    Stack(size_t size) : size(size) { stack = (T *)malloc(size * sizeof(T)); }
    bool push(T item) {
        if (top >= (int)(size - 1)) {
            T *tmp = (T *)realloc(stack, (size + increment) * sizeof(T));
            if (!tmp)
                return false;
            stack = tmp;
            size += increment;
        }
        stack[++top] = item;
        return true;
    }
    bool pop(T &item) {
        if (top < 0)
            return false;
        item = stack[top--];
        if (top != -1 && top < (int)(size - increment)) {
            T *tmp = (T *)realloc(stack, (size - increment) * sizeof(T));
            if (!tmp)
                return false;
            stack = tmp;
            size -= increment;
        }
        return true;
    }
};
```

2.

```
template <class T> bool peek(Stack<T> s, T &el) {  
    if (!s.pop(el))  
        return 0;  
    s.push(el);  
    return true;  
};
```

3.

//funkciju dodati u klasu Stack iz prvog zadatka te tek tada pokrenuti

```
bool peek(int &el, int index) {  
    Stack<T> pom;  
    int i;  
    T j;  
    bool found;  
    found = false;  
    i = -1;  
    while (this->pop(j)) {  
        pom.push(j);  
        i++;  
        if (i == index) {  
            found = true;  
            el = j;  
            break;  
        }  
    }  
    while (pom.pop(j))  
        this->push(j);  
    return found;  
}
```

4.

```
template <class T> class Queue {
private:
    static const int MAX = 100;
    T queue[MAX];
    int write = 0;
    int read = 0;

public:
    bool enqueue(T item) {
        if ((write + 1) % MAX == read)
            return false;
        queue[write] = item;
        write = (write + 1) % MAX;
        return true;
    }
    bool dequeue(T &item) {
        if (write == read)
            return false;
        item = queue[read];
        read = (read + 1) % MAX;
        return true;
    }
    int count() {
        if (write >= read) {
            return (write - read);
        } else {
            return (write - read + MAX);
        }
    }
    bool peek(int &el) {
        Queue<T> pom;
        T i;
        if (!this->dequeue(i))
            return 0;
        el = i;
        pom.enqueue(i);
        while (this->dequeue(i))
            pom.enqueue(i);
        while (pom.dequeue(i))
            this->enqueue(i);
        return true;
    }
};
```

5.

//funkciju dodati u klasu Queue iz četvrtog zadatka te tek tada pokrenuti

```
int peek(int &el, int index) {
    Queue<T> pom;
    T i;
    int j;
    bool found = false;
    j = -1;
    if (!this->dequeue(i))
        return 0;
    j++;
    if (j == index) {
        el = i;
        found = 1;
    }
    pom.enqueue(i);
    while (this->dequeue(i)) {
        j++;
        if (j == index) {
            el = i;
            found = 1;
        }
        pom.enqueue(i);
    }
    while (pom.dequeue(i))
        this->enqueue(i);
    return found;
}
```

6.

```
template <class T> bool
transfer(Stack<T> *s, Queue<T> *q) {
    T element;
    bool status;
    status = s->pop(element);
    if (status == false)
        return true;
    status = q->enqueue(element);
    if (status == false)
        return false;
    return transfer(s, q);
}
```

```
int main(void) {
    Queue<int> q1;
    Stack<int> s1;
    int i;
    for (i = 0; i <= 10; i++)
        s1.push(i);
    transfer(&s1, &q1);
    return 0;
}
```

S

10 → S = true, e = 10 → 10 | s = 1
 9 → s = +1 e = 9 → 10, 9 | s = 1
 8
 7

7, 7
 10, 9, 8, 7

7.

```
template <class T> Queue<T> *selectMultiples(Queue<T> *q, int number) {
    Queue<T> *tmp = new Queue<T>();
    Queue<T> *tmp2 = new Queue<T>();
    int element, temp;
    while (q->dequeue(element)) {
        if (element % number == 0)
            tmp->enqueue(element);
        else
            tmp2->enqueue(element);
    }
    while (tmp2->dequeue(element)) {
        q->enqueue(element);
    }
    delete tmp2;
    return tmp;
}

int main(void) {
    //... inicijalizacija i punjenje q1
    Queue<int> *q3 = selectMultiples(&q1, 2);
    return 0;
}
```

8.

```
template <class T> void remove(Stack<T> *s) {
    Stack<T> s1, s2;
    int el, x;
    while (s->pop(el)) {
        s1.push(el);
        while (s->pop(x))
            if (x != el)
                s2.push(x);
        while (s2.pop(x))
            s->push(x);
    }
    while (s1.pop(el))
        s->push(el);
}

int main(void) {
    Queue<int> q;
    //... inicijalizacija i punjenje s1
    remove(&s1);
    return 0;
}
```