Algoritmi i strukture podataka 2019./2020.

Zadatci za vježbu – liste

U zadatcima se koristi razred `List` te pomoćni razredi `ListElement` i `ListIterator`:

```
template <typename T> struct ListElement {
	T data;
	ListElement<T> *next;
};

template <typename T> class ListIterator {
private:
	ListElement<T> *ptr = nullptr;

public:
	ListIterator<T>() : ptr(nullptr) {}
	ListIterator<T>(ListElement<T> *ptr) : ptr(ptr) {}
	ListIterator<T> &operator++(); // prefix ++
	ListIterator<T> operator++(int);// postfix ++
	bool operator!=(const ListIterator<T> &other) { return ptr != other.ptr; }
	T &operator*() { return ptr->data; }
};


template <typename T>
class List {
	ListElement<T> *head = nullptr;
public:
	typedef ListIterator<T> iterator;
	iterator begin() { return iterator(head); }
	iterator end() { return iterator(); }

	bool insert(T data); // dodavanje na početak liste
	bool append(T data); // dodavanje na kraj liste
	bool insertSorted(T data); // umetanje po sortiranom redoslijedu
	bool remove(T data);
	void print();
};
```

Neka je u glavnom programu lista definirana i napunjena elementima na sljedeći način:

```
List<int> l;
l.append(4);
l.append(5);
l.insert(1);
l.insert(2);
l.insert(3);
```

1. Napišite funkcijski član razreda `List` koji vraća referencu na vrijednost pohranjenu u *k*-tom elementu liste, a zadan je prototipom:
   `T& at(int k);`

Funkcijski član vraća referencu na vrijednost pohranjenu u *k*-tom članu liste počevši s indeksom 0. Koristiti iterator.
Napomena: ako indeks nije dobro zadan (k < 0 ili k >= broju članova liste), onda treba baciti odgovarajuću iznimku. Npr. za `l.getElem(3)` funkcijski član treba vratiti 4.

2. Napišite funkcijski član razreda `List` koji vraća pokazivač na *k*-ti element liste, a zadan je prototipom:
```
ListElement<T>* at2(int k);
```
Ako indeks nije dobro zadan (k < 0 ili k >= broju članova liste), onda treba vratiti nullptr. Npr. za `l->getElem(3)` funkcijski član treba vratiti 4.

3. Napišite funkcijski član razreda `List` koja određuje i vraća duljinu liste, tj. broj elemenata u listi, a zadan je prototipom:
```
int length();
```

4. Napišite funkcijski član razreda `List` koja određuje i vraća broj pojavljivanja zadane vrijednosti `item` u listi, a zadana je prototipom:
```
int countItem(T item);
```

5. Napišite funkcijski član razreda `List` koja briše sve elemente liste:
```
void deleteAll();
```

6. Napisati nerekurzivni funkcijski član razreda `List` koja će promjenom pokazivača na sljedeće elemente obrnuti listu (tako da po izlasku iz funkcijskog člana glava liste pokazuje na čvor koji je inicijalno bio zadnji u listi), a čiji je prototip:
```
void reverseList();
```

7. (prema zadatku za ljetni ispitni rok 2012./2013.)
Napisati funkcijski član koja će omogućiti **rekurzivno** obrtanje liste promjenom pokazivača na sljedeće elemente (tako da po izlasku iz funkcijskog člana glava liste pokazuje na čvor koji je inicijalno bio zadnji u listi), a čiji je prototip:
```
void reverseListR();
```
Napomena: koristite pomoćni funkcijski član, npr.
```
void reverseListRAux(ListElement<T> **p);
```

8. Napišite funkciju koja briše iz uzlazno poredane liste sve čvorove koji sadrže duplikate te vraća broj obrisanih čvorova:
```
int removeDuplicates();
```

9. (prema zadatku sa završnog ispita 2014./2015.)
Neka je u glavnom programu lista definirana kao: `List<Student> l;`
Razred `Student` definiran je na sljedeći način:
```
struct Student {
  string imePrezime;
  char spol;
  int dob;
};
```

Napišite funkcijski član prototipa:

```
void List<Student>::rearrangeList();
```

koja će elemente liste (zapise o studentima) rasporediti ovisno o spolu. U novoj listi, zapisi o osobama ženskog spola (spol 'Z') trebaju se nalaziti na početku liste, a zapisi o osobama muškog spola (spol 'M') na kraju. Funkcija ne smije alocirati nove čvorove.

Primjer:

Listu (od početka prema kraju):

```
("Ivo",'M',19);("Ana",'Z',18);("Mara",'Z',18);("Mate",'M',19);
```

funkcija transformira u:

```
("Ana",'Z',18);("Mara",'Z',17);("Ivo",'M',19);("Mate",'M',19);
```

10. (prema zadatku s ispitnog roka)
    Napisati funkcijski član koji određuje jednostruko povezanu listu u kojoj je pohranjen slijed poteza koji rješavaju problem Hanojskih tornjeva za **n** diskova, jedan potez po elementu liste. Neka je u glavnom programu lista definirana kao: `List<Potez> l;`
    Razred `Potez` definiran je na sljedeći način:

```
struct Potez {
   int n;
   char t1;
   char t2;
};
```

    gdje je **n** redni broj diska, **t1** oznaka tornja s kojeg se disk uzima a **t2** oznaka tornja na koji se disk stavlja. Prototip je:

```
void hanoi(char izvor, char odrediste, char pomocni, int n);
```

11. Napisati glavni program/glavne programe u kojima ćete provjeriti napisane funkcijske članove iz zadataka 1.-10.

Rješenja

1.

```
template <typename T>
T& List<T>::at(int k) { // dohvati ref. na podatak u k-tom elementu liste
	int i;
	List<T>::iterator it;
	for (i = 0, it = this->begin(); i < k && it != this->end(); i++, it++) ;
	if (k >= 0 && it != this->end()) return *it;
	else throw std::runtime_error("Index out of range.");
}
```

2.

```
template <typename T>
ListElement<T>* List<T>::at2(int k) { // dohvati pokazivač na k-ti element liste
	if (k < 0) return nullptr;
	int i;
	ListElement<T> *p;
	for (i = 0, p = head; i < k && p; i++, p = p->next);
	return p;
}
```

3.

```
template <typename T>
int List<T>::length() {
	int len = 0;
	for (iterator it = this->begin(); it != this->end(); len++, it++);
	return len;
}
```

4.

```
template <typename T>
int List<T>::countItem(T item) {
	int cnt = 0;
	for (List<T>::iterator it = this->begin(); it != this->end(); it++) {
		if (*it == item) ++cnt; // operator == treba biti preopterećen za T
	}
	return cnt;
}
```

5.

```
template <typename T>
void List<T>::deleteAll() {
	ListElement<T> *p = head;
	while (p) {
		ListElement<T> *tmp = p;
		p = p->next;
		delete tmp;
	}
	head = nullptr;
}
```

6.

```
template <typename T>
void List<T>::reverseList() {
	ListElement<T> *p = head;
	ListElement<T> *pprev = nullptr;
	ListElement<T> *pnext = nullptr;

	while (p) {
		pnext = p->next;
		p->next = pprev;
		pprev = p;
		p = pnext;
	}
	head = pprev;
}
```

7.

```
template <typename T>
void List<T>::reverseListRAux(ListElement<T> **p) {
	ListElement<T> *first = nullptr;
	ListElement<T> *fnext = nullptr;

	if (!*p) return; // prazna lista
	first = *p;
	fnext = first->next;
	if (!fnext) return;

	reverseListRAux(&fnext);
	first->next->next = first;
	first->next = nullptr;

	*p = fnext;
}

template <typename T>
void List<T>::reverseListR() {
	reverseListRAux(&head);
}
```

8.

```
template <typename T>
int List<T>::removeDuplicates() {
	ListElement<T> *p = head;
	int cnt = 0;
	while (p) {
		ListElement<T> *pnext = p->next;
		if (pnext && pnext->data == p->data) {
			p->next = p->next->next;
			delete pnext;
			++cnt;
		}
		p = p->next;
	}
	return cnt;
}
```

9.

```
void List<Student>::rearrangeList() {
	ListElement<Student> *firstF = nullptr, *firstM = nullptr;
	ListElement<Student> *lastF = nullptr, *lastM = nullptr;

	ListElement<Student> *p = head, *pnext = nullptr;
	while (p) {
		pnext = p->next;
		if (p->data.spol == 'Z') {
			if (!firstF) {
				firstF = lastF = p;
			}
			lastF->next = p;
			lastF = p;
		}
		else if (p->data.spol == 'M') {
			if (!firstM) {
				firstM = lastM = p;
			}
			lastM->next = p;
			lastM = p;
		}
		p = pnext;
	}
	lastF->next = firstM;
	head = firstF;

}
```

10.

```
void List<Potez>::hanoi(char izvor, char odrediste, char pomocni, int n) {
	if (n > 0) {
		hanoi(izvor, pomocni, odrediste, n - 1);
		// Stvaramo novi element
		this->append({ n, izvor, odrediste });
		hanoi(pomocni, odrediste, izvor, n - 1);
	}
}
```