Napredno programiranje i programski jezici

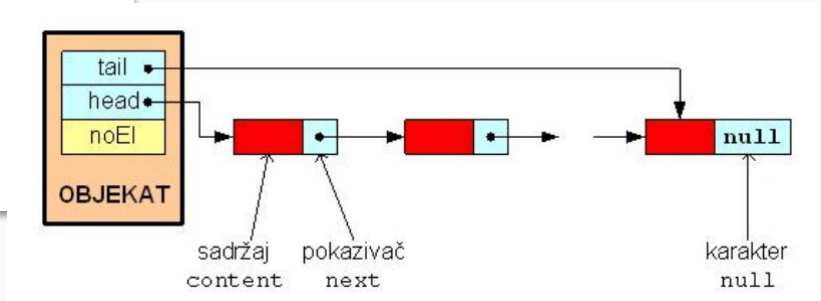
05 C++ (lista)

Fakultet tehničkih nauka, Novi Sad
23-24/Z
Dunja Vrbaški

ZADATAK

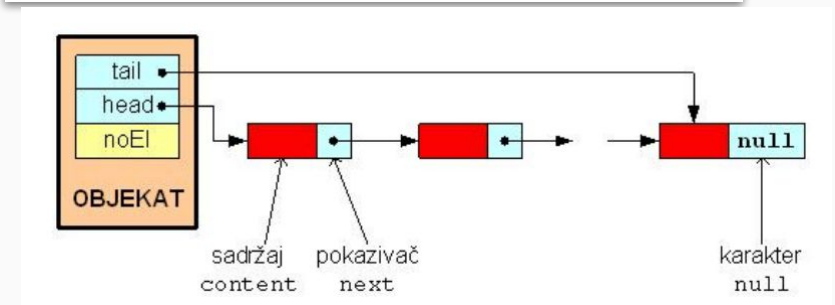
Realizovati jednostruko spregnutu listu proizvoljnog tipa.

```
template <class T>
class List {
private:
    struct listEl{
        T content;
        struct listEl *next;
    };
    listEl *head;
    listEl *tail;
    int noEl;

public:
   ...
};
```



```
template <class T>
class List {
private:
      struct listEl{
            T content;
            struct listEl *next;
      };
      listEl *head;
      listEl *tail;
      int noEl;

public:
   ...
};
```

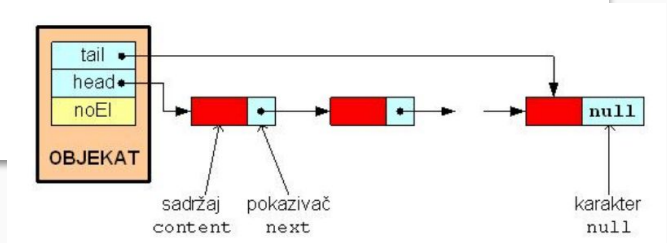
```
int main() {

    List<int> iList;

    iList.add(1,5);
    iList.add(2,7);
    cout << iList << endl;

    iList.remove(1);
    cout << iList << endl;

}
```



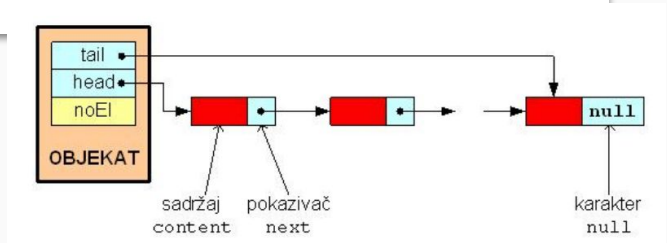
```
template <class T>
class List {
private:
      struct listEl{
            T content;
            struct listEl *next;
      };
      listEl *head;
      listEl *tail;
      int noEl;

public:
      List(){
            head = tail = NULL;
            noEl = 0;
      }
      List(const List<T>&);
      virtual ~List();

      List<T>& operator=(const List<T>&);
```

```
      int size() const { return noEl; }
      bool empty() const {
            return head == NULL ? 1 : 0;
      }

      bool add(int, const T&);
      bool remove(int);
      bool read(int, T&)const;
      void clear();
};
```

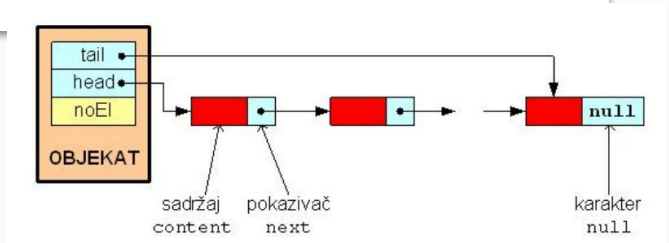
```
template <class T>
bool List<T>::read(int n,T& retVal) const {
     if(n < 1 || n > noEl)
          return false;
     if(n == 1)
          retVal = head->content;
     else if(n == noEl)
          retVal = tail->content;
     else {
          listEl* temp = head;
          for(int i = 1; i < n; i++)
               temp = temp->next;
          retVal = temp->content;
     }
     return true;
}
```



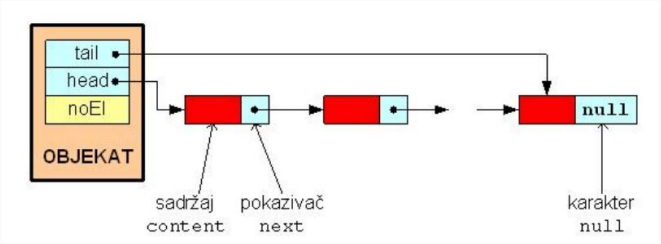
```
template <class T>
List<T>::List(const List<T>& rl) {
     head = NULL;
     tail = NULL;
     noEl = 0;

     for(int i = 1; i <= rl.noEl; i++){
          T res;
          if(rl.read(i, res))
               add(i, res);
     }
}
```



```
ostream& operator<<(ostream& out, const List<T>& rl) {
     out << endl;
     out << "--------" << endl;
     for(int i = 1; i <= rl.size(); i++){
          if(i != 1) out << ", ";
          T res;
          rl.read(i, res);
          out << res;
     }
     out << endl << "--------" << endl;
     return out;
}
```



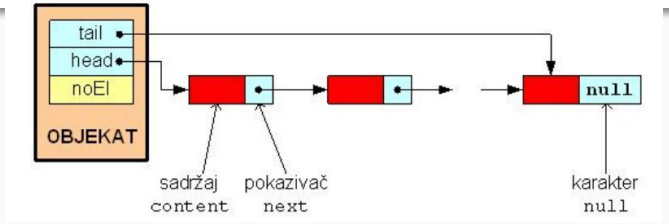
```
template <class T>
bool List<T>::add(int n, const T& newContent)
{
	if(n < 1 || n > noEl + 1)
		return false;

	listEl* newEl = new listEl;
	if(newEl == NULL)
		return false;

	newEl->content = newContent;
```



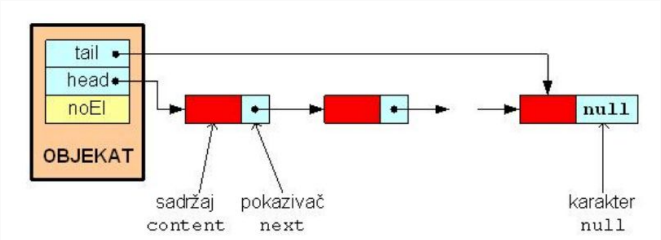

```
	if(n == 1) {
		newEl->next = head;
		head = newEl;
	} else if(n == noEl + 1) {
		newEl->next = NULL;
		tail->next = newEl;
	} else {
		listEl* temp = head;
		for(int i = 2; i < n; i++)
			temp = temp->next;
		newEl->next = temp->next;
		temp->next = newEl;
	}
	noEl++;
	if(newEl->next == NULL)
		tail = newEl;
	return true;
}
```

```
template <class T>
bool List<T>::remove(int n){
	if(n < 1 || n > noEl)
		return false;

	if(n == 1) {
		listEl* del = head;
		head = head->next;
		if(tail == del)
			tail = NULL;
		delete del;
		noEl--;
	}
```

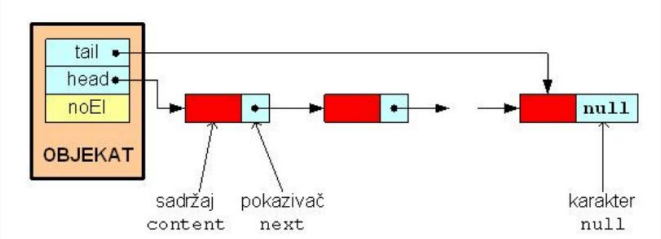
```
	else {
		listEl* temp = head;
		for(int i = 2; i < n; i++)
			temp = temp->next;
		listEl* del = temp->next;
		temp->next = del->next;
		if(tail == del)
			tail = temp;
		delete del;
		noEl--;
	}
	return true;
}
```



```
template <class T>
void List<T>::clear() {
     while(!empty())
          remove(1);
}
```



```
template <class T>
List<T>::~List() {
     while(!empty())
          remove(1);
}
```



```
template <class T>
List<T>& List<T>::operator=(const List<T>& rl) {
     if(this != &rl) {
          clear();

          head = NULL;
          tail = NULL;
          noEl = 0;

          for(int i = 1; i <= rl.noEl; i++){
               T res;
               if(rl.read(i, res))
                    add(i, res);
          }
     }
     return *this;
}
```