# STRUKTURE PODATAKA I ALGORITMI

ZBIRKA ZADATAKA

Ishod učenja 2

2024-2025.

## Zadatak 1

Napišite program koji radi sljedeće:

- Napravite jednostruku povezanu listu tako da sadrži brojeve iz raspona [1, 10].
- Napravite dvostruko povezanu listu tako da sadrži brojeve iz raspona [11, 20].
- Na najefikasniji mogući način napravite vektor koji prvo sadrži kopirane vrijednosti iz jednostruko povezane liste, a zatim iz dvostruko povezane liste. Na kraju ovog koraka vektor treba sadržavati brojeve iz raspona [1, 20].
- Pitajte korisnika da unese jedan broj između 0 i 19, a vi zatim na tu poziciju u vektor umetnite vrijednost 99.
- Obrnite sadržaj vektora (primjerice, ako vektor sadrži 99, 1, 2, 3, ..., 20, nakon obrtanja treba sadržavati 20, ..., 3, 2, 1, 99).
- Prekopirajte sadržaj vektora u stog.
- Prekopirajte sadržaj stoga u red.
- Ispišite sadržaj reda.

*Moguće rješenje:*

```
#include <fstream>
#include <iostream>
#include <forward_list>
#include <list>
#include <string>
#include <vector>
#include <stack>
#include <queue>
using namespace std;

int main() {
	// Napravite jednostruku povezanu listu tako da sadrži brojeve iz raspona[1, 10].
	forward_list<int> sll;
	for (int i = 10; i > 0; i--) {
		sll.push_front(i);
	}

	// Napravite dvostruko povezanu listu tako da sadrži brojeve iz raspona [11, 20].
	list<int> dll;
	for (int i = 11; i <= 20; i++) {
		dll.push_back(i);
	}

	// Na najefikasniji mogući način napravite vektor koji prvo sadrži kopirane vrijednosti iz jednostruko povezane liste, a zatim iz dvostruko povezane liste. Na kraju ovog koraka vektor treba sadržavati brojeve iz raspona [1, 20].
	vector<int> vec(sll.begin(), sll.end());
	vec.insert(vec.end(), dll.begin(), dll.end());

	// Pitajte korisnika da unese jedan broj između 0 i 19, a vi zatim na tu poziciju u vektor umetnite vrijednost 99.
	int pos;
	cout << "Unesite poziciju (0-19): ";
	cin >> pos;
	vec.insert(vec.begin() + pos, 99);

	// Obrnite sadržaj vektora (primjerice, ako vektor sadrži 99, 1, 2, 3, …, 20, nakon obrtanja treba sadržavati 20, …, 3, 2, 1, 99).
	reverse(vec.begin(), vec.end());
```

```
	// Prekopirajte sadržaj vektora u stog.
	stack<int> st;
	for (int i = 0; i < vec.size(); i++) {
		st.push(vec[i]);
	}

	// Prekopirajte sadržaj stoga u red.
	queue<int> q;
	while (!st.empty()) {
		q.push(st.top());
		st.pop();
	}

	// Ispišite sadržaj reda.
	while (!q.empty()) {
		cout << q.front() << " ";
		q.pop();
	}
}
```

## Zadatak 2

Datoteka **Genome_Vibrio_cholerae.txt** sadrži nukleotide koji čine DNA poznate bakterije. Napišite program koji čita svaki znak iz datoteke te ga stavlja u red. Nakon što ste sve znakove stavili u red, izračunajte vrijednost varijable odnos_cg na sljedeći način:

- Postavite odnos_cg na nulu.
- Svaki put kad u redu naiđete na znak C (citozin), smanjite odnos_cg za 1.
- Svaki put kad u redu naiđete na znak G (gvanin), povećajte odnos_cg za 1.
- Kad u redu naiđete na A (adenin) ili T (timin), nemojte mijenjati vrijednost varijable odnos_cg.

Na kraju ispišite vrijednost varijable odnos_cg.

*Moguće rješenje:*

```
#include <fstream>
#include <iostream>
#include <queue>
using namespace std;

int main() {
	ifstream fin("Genome_Vibrio_cholerae.txt");
	if (!fin) {
		cout << "error while opening file" << endl;
		return -1;
	}

	queue<char> q;
	char c;
	while (fin >> c) {
		q.push(c);
	}

	int odnos_cg = 0;
	while (!q.empty()) {
		if (q.front() == 'C') {
			odnos_cg--;
		}
		else if (q.front() == 'G') {
			odnos_cg++;
		}
		q.pop();
	}

	cout << odnos_cg << endl;
}
```

## Zadatak 3

Napišite program koji korisniku omogućuje upis slova, pri čemu sljedeća slova imaju posebno značenje:

- Kad korisnik upiše X, unos je gotov i trebate mu ispisati finalni uneseni tekst.
- Kad korisnik upiše Z, želi odustati (engl. *undo*) od prethodno upisanog slova. Ako nema slova za *undo*, program mora ignorirati naredbu i nastaviti raditi.
- Kad korisnik upiše Y, ipak želi vratiti (engl. *redo*) prethodno upisano slovo od kojeg je odustao. Ako nema slova za *redo*, program mora ignorirati naredbu i nastaviti raditi.

Koristite potreban broj stogova, bez ikakvih drugih kontejnera. U nastavku slijedi nekoliko primjera rada programa:

| Korisnik upisuje slova bez *undo*/*redo*: | Korisnik pogriješi u jednom slovu pa napravi *undo*: | Korisnik se predomisli pa napravi nekoliko *undo*: | Korisnik se predomisli dva puta: |
|---|---|---|---|
| > g<br>> a<br>> r<br>> f<br>> i<br>> e<br>> l<br>> d<br>> X<br>garfield | > g<br>> a<br>> r<br>> m<br>> Z<br>> f<br>> i<br>> e<br>> l<br>> d<br>> X<br>garfield | > g<br>> a<br>> r<br>> Z<br>> Z<br>> Z<br>> Z<br>> o<br>> d<br>> i<br>> e<br>> X<br>odie | > o<br>> d<br>> i<br>> Z<br>> Z<br>> Z<br>> Y<br>> Y<br>> Y<br>> Y<br>> e<br>> X<br>odie |

*Moguće rješenje:*

```
#include <iostream>
#include <stack>
using namespace std;

int main() {
	stack<char> stack_main;
	char c;

	stack<char> stack_undo;
	bool dalje = true;
	while (dalje) {
		cout << "> ";
		cin >> c;
		switch (c) {
		case 'Z': // undo
			if (!stack_main.empty()) {
				stack_undo.push(stack_main.top());
				stack_main.pop();
			}
			break;
		case 'Y': // redo
			if (!stack_undo.empty()) {
				stack_main.push(stack_undo.top());
				stack_undo.pop();
			}
			break;
		case 'X': dalje = false; break;
		default: stack_main.push(c);
		}
	}
```

```
	stack<char > s;
	while (!stack_main.empty()) {
		s.push(stack_main.top());
		stack_main.pop();
	}

	while (!s.empty()) {
		cout << s.top();
		s.pop();
	}
}
```

## Zadatak 4

Stvorite vektor od 10 nasumičnih brojeva u rasponu [1, 1000] i pripremite praznu dvostruko povezanu listu. Iterirajte po vektoru tako da u svakom prolazu pronađete najmanji element u vektoru pa ga prebacite u listu (kad pronađete najmanji element u vektoru, kopirajte ga u listu, a zatim uklonite iz vektora). Ponavljajte postupak dok vektor ne ostane prazan. Ispišite sve elemente iz liste od kraja prema početku.

*Moguće rješenje:*

```
#include <ctime>
#include <iostream>
#include <list>
#include <vector>
using namespace std;

int generate_random(int min, int max) {
	return rand() % (max - min + 1) + min;
}

int main() {

	srand(time(nullptr));

	vector<int> numbers;

	for (int i = 0; i < 10; i++) {
		numbers.push_back(generate_random(1, 1000));
	}

	list<int> l;

	while (!numbers.empty()) {
		vector<int>::iterator smallest = numbers.begin();
		for (auto it = numbers.begin() + 1; it != numbers.end(); ++it)
		{
			if (*it < *smallest) {
				smallest = it;
			}
		}

		l.push_back(*smallest);

		numbers.erase(smallest);
	}

	for (auto it = l.rbegin(); it != l.rend(); ++it)
	{
		cout << *it << " ";
	}
	cout << endl;
}
```

## Zadatak 5

Pročitajte sve retke iz datoteke **input.txt** i pohranite ih u dvostruko povezanu listu tako da elemente ubacujete na početak. Zatim kopirajte sve elemente iz povezane liste u vektor na najefikasniji mogući način. Nakon toga, prođite kroz vektor unatrag i svaki element kopirajte u red. Prekopirajte podatke iz reda u datoteku.

Izmjerite trajanje postupka upisa u datoteku.

*Moguće rješenje:*

```
#include <chrono>
#include <fstream>
#include <iostream>
#include <list>
#include <queue>
#include <stack>
#include <string>
#include <vector>
using namespace std;

void write_to_file(string path, queue<string>& q) {
	ofstream fout(path);
	if (!fout) {
		cout << "error while opening output file" << endl;
		return;
	}

	while (!q.empty()) {
		fout << q.front() << endl;
		q.pop();
	}

	fout.close();
}

int main() {

	ifstream fin("input.txt");
	if (!fin) {
		cout << "error while opening file" << endl;
		return -1;
	}

	list<string> lines;
	string line;
	while (getline(fin, line)) {
		lines.push_front(line);
	}

	fin.close();

	vector<string> vecCopy(lines.begin(), lines.end());

	queue<string> q;

	for (auto it = vecCopy.rbegin(); it < vecCopy.rend(); ++it) {
```

```
            q.push(*it);
        }

        auto start = chrono::high_resolution_clock::now();

        write_to_file("output.txt", q);

        auto end = chrono::high_resolution_clock::now();
        auto duration = chrono::duration_cast<chrono::microseconds>(end -
start).count();

        cout << "finished in " << duration << " microseconds" << endl;
}
```

## Zadatak 6

Simulirajte rad štoperice tako što ćete prvo postaviti početnu vremensku točku na početku programa. Zatim omogućite korisniku da napravi kontrolnu točku kad god želi. Izračunajte i ispišite vrijeme proteklo između svake kontrolne točke i početne točke. Sve korisnički definirane kontrolne točke čuvajte u redu.

*Moguće rješenje:*

```
#include <chrono>
#include <iostream>
#include <queue>
using namespace std;
using namespace std::chrono;

int main() {

	queue<time_point<high_resolution_clock>> points;

	auto start = high_resolution_clock::now();

	while (true) {
		cout << "make a checkpoint? (y - yes, q - quit): ";
		char op;
		cin >> op;

		if (op == 'y') {
			auto point = high_resolution_clock::now();
			points.push(point);
		}
		else if (op == 'q') {
			break;
		}
		else {
			cout << "invalid operation!" << endl;
		}
	}

	int i = 0;
	while (!points.empty()) {
		auto duration = duration_cast<milliseconds>(points.front() -
start).count();

		points.pop();

		cout << "#" << ++i << " lap: " << duration << " ms" << endl;
	}
}
```

## Zadatak 7

Napišite program koji provjerava je li dana HTML datoteka balansirana. Svaka početna oznaka (npr. <head>) mora imati odgovarajuću završnu oznaku (npr. </head>) te pazite na ugnježđivanje (npr.  je u redu, dok  nije). Pretpostavite da u dokumentu neće biti dodatnog teksta već samo HTML tagovi bez atributa. Upotrijebite odgovarajuću strukturu podataka za rješavanje ovog problema.

*Moguće rješenje:*

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <stack>
using namespace std;

string extract_end_tag(ifstream& fin) {
	stringstream sout;
	char token;
	while (fin >> token) {
		if (token == '>')
			break;
		sout << token;
	}

	return sout.str();
}

string extract_start_tag(char first_letter, ifstream& fin) {
	return first_letter + extract_end_tag(fin);
}

int main() {
	ifstream fin("source.html");
	if (!fin) {
		cout << "error: cannot open file" << endl;
		return -1;
	}

	stack<string> tags;

	char token;
	string tag;

	while (fin >> token) {
		if (token == '<') {
			fin >> token;
			if (token == '/') { // closing tag
				tag = extract_end_tag(fin);
				if (tag == tags.top()) {
					tags.pop();
				}
				else {
					cout << "error: no closing tag for <" << tags.top()
<< ">" << endl;
					return -1;
				}
			}
```

```
                        else { // opening tag
                              tag = extract_start_tag(token, fin);
                              tags.push(tag);
                        }
                        //cout << tag << endl;
                  }
            }
            fin.close();

            if (!tags.empty()) {
                  cout << "error: unbalanced html file" << endl;
            }
            else {
                  cout << "file is balanced" << endl;
            }
}
```

## Zadatak 8

Pročitajte u red sve podatke o paketima pohranjene u datoteci **traffic.csv**. Svaki paket opisan je izvornom i odredišnom IP adresom te veličinom u bajtovima. Potrebno je implementirati novi tip podataka koji će pratiti koliko je bajtova primila svaka odredišna IP adresa, uz mehanizam za povećanje broja primljenih bajtova za određenu adresu i dohvaćanje tog broja.

Omogućite korisniku da unese graničnu vrijednost (threshold), a zatim obradite svaki paket iz datoteke tako da za svaku odredišnu adresa pamtite primljenu količinu bajtova. Ako odredišna adresa treba primiti više bajtova od definirane granične vrijednosti, odbacite paket (nemojte dalje zbrajati bajtove za to odredište) i ispišite to u konzolu. Na kraju ispišite koliko bajtova je primila koja odredišna adresa.

*Moguće rješenje:*

DestinationDictionary.h

```
#pragma once
#include <string>
#include <vector>
#include "Destination.h"
using namespace std;

class DestinationDictionary {
private:
	vector<Destination> ip_addrs;
	int find_index_of(string addr);

public:
	int read_bytes(string addr);
	void store_bytes(string addr, unsigned size);
	void print();
};
```

DestinationDictionary.cpp

```
#include <algorithm>
#include <iostream>
#include "DestinationDictionary.h"

int DestinationDictionary::read_bytes(string addr)
{
	int idx = find_index_of(addr);
	if (idx == -1) return 0;

	return ip_addrs[idx].size;
}

void DestinationDictionary::store_bytes(string addr, unsigned size)
{
	int idx = find_index_of(addr);
	if (idx == -1) {
		ip_addrs.emplace_back(addr, size);
	}
	else {
		ip_addrs[idx].size += size;
	}
```

```
}

void DestinationDictionary::print()
{
	cout << "\nFinal amout of received bytes:" << endl;
	for (int i = 0; i < ip_addrs.size(); i++)
	{
		cout << ip_addrs[i].dest << ": " << ip_addrs[i].size << " bytes" <<
endl;
	}
}

int DestinationDictionary::find_index_of(string addr)
{
	for (int i = 0; i < ip_addrs.size(); i++)
	{
		if (ip_addrs[i].dest == addr) {
			return i;
		}
	}

	return -1;
}
```

Packet.h

```
#pragma once
#include <string>
using namespace std;

struct Packet {
	string src;
	string dest;
	unsigned size;

	Packet(string src, string dest, unsigned size);
};
```

Packet.cpp

```
#include "Packet.h"

Packet::Packet(string src, string dest, unsigned size)
	: src(src), dest(dest), size(size) {
}
```

Destination.h

```
#pragma once
#include <string>
using namespace std;

struct Destination {
	string dest;
	unsigned size;

	Destination(string dest, unsigned size);
};
```

Destination.cpp

```
#include "Destination.h"

Destination::Destination(string dest, unsigned size)
	: dest(dest), size(size) {
}
```

Source.cpp

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <queue>
#include "DestinationDictionary.h"
#include "Packet.h"
using namespace std;

int main() {
	ifstream fin("traffic.csv");
	if (!fin) {
		cout << "error" << endl;
		return -1;
	}

	queue<Packet> packets;
	string line;
	getline(fin, line); // skip header

	while (getline(fin, line)) {
		stringstream sstr(line);
		string src, dst;
		unsigned size;

		getline(sstr, src, ';');
		getline(sstr, dst, ';');
		sstr >> size;

		packets.emplace(src, dst, size);
	}

	fin.close();

	DestinationDictionary dict;

	unsigned threshold;
	cout << "threshold: ";
	cin >> threshold;

	while (!packets.empty()) {
		Packet packet = packets.front();

		unsigned size = dict.read_bytes(packet.dest);
		if (packet.size + size > threshold) {
			cout << packet.src << " -> " << packet.dest << " (" <<
packet.size << " B) dropped\n";
```

```
                }
                else {
                        dict.store_bytes(packet.dest, packet.size);
                }

                packets.pop();
        }

        dict.print();
}
```

## Zadatak 9

Učitajte sve retke iz datoteke **input6.txt** u dvostruko povezanu listu. Implementirajte jednostavni tekstualni editor koji označava odabrani red (počnite s prvim redom) i omogućuje korisniku:

- Prijelaz na sljedeći red (**n**)
- Prijelaz na prethodni red (**b**)
- Dodavanje novog retka **nakon** trenutno odabranog i odmah prijelaz na taj redak (**e**)
- Brisanje trenutno odabranog retka (**x**)
- Spremanje natrag u istu datoteku (**s**)
- Izlaz (**q**)

Nakon svake operacije ispišite sadržaj datoteke. Primjer rada programa:

```
> Line one.
  Line two.
  Line three.

(n, b, e, x, s, q): n

  Line one.
> Line two.
  Line three.

(n, b, e, x, s, q): e

: NEW LINE 1

  Line one.
  Line two.
> NEW LINE 1
  Line three.

(n, b, e, x, s, q): b

  Line one.
> Line two.
  NEW LINE 1
  Line three.

(n, b, e, x, s, q): e

: NEW LINE 2

  Line one.
  Line two.
> NEW LINE 2
  NEW LINE 1
  Line three.

(n, b, e, x, s, q): x

  Line one.
  Line two.
> NEW LINE 1
  Line three.
```

*Moguće rješenje:*

```
#include <fstream>
#include <iostream>
#include <list>
#include <sstream>
#include <string>
using namespace std;

void print(list<string>& buffer, list<string>::iterator& curr) {
	for (auto it = buffer.begin(); it != buffer.end(); ++it)
	{
		if (curr == it) {
			cout << "> ";
		}
		else {
			cout << "  ";
		}

		cout << *it << endl;
	}
}

void flush(list<string>& buffer, string filepath) {
	ofstream fout(filepath);
	for (auto it = buffer.begin(); it != buffer.end(); ++it)
	{
		fout << *it << endl;
	}
	fout.close();
}

int main() {
	// učitavanje iz datoteke
	ifstream fin("input6.txt");
	if (!fin) {
		cout << "error opening file" << endl;
		return -1;
	}

	list<string> lines;
	string line;
	while (getline(fin, line)) {
		lines.push_back(line);
	}
	fin.close();

	auto curr = lines.begin();
	while (true) {
		print(lines, curr);
		cout << "\n(n, b, e, x, s, q): ";
		char op;
		cin >> op;

		if (op == 'n') {
			curr++;
			if (curr == lines.end()) curr--;
		}
```

```
            else if (op == 'b') {
                  if (curr != lines.begin()) curr--;
            }
            else if (op == 'e') {
                  cout << "\n: ";
                  string line;
                  cin.ignore();
                  getline(cin, line);
                  if (curr != lines.end()) curr++;
                  lines.insert(curr, line);
                  --curr;
            }
            else if (op == 'x') {
                  if (lines.empty()) continue;
                  curr = lines.erase(curr);
                  if (curr == lines.end() and curr != lines.begin()) curr--;
            }
            else if (op == 's') {
                  flush(lines, "input6.txt");
            }
            else if (op == 'q') {
                  break;
            }

            cout << endl;
      }

      return 0;
}
```

## Zadatak 10

Implementirajte **Shunting yard** algoritam koji pretvara **infiksni izraz** (npr. 3 + 4 * 2 / 1 - 5) u **postfiksni zapis** (obrnuta poljska notacija). Omogućite korisniku unos infiksnog izraza, a zatim ispišite postfiksnu verziju tog izraza. Pretpostavite da će korisnik koristiti samo operacije +, -, * i / te da neće koristiti zagrade.

Infiks izraz: 3 + 4 * 2 / 1 - 5
Postfiks izraz: 3 4 2 * 1 / + 5 -

*Moguće rješenje:*

```
#include <iostream>
#include <queue>
#include <sstream>
#include <stack>
#include <string>
using namespace std;

bool is_op(char op) {
	return op == '+' or op == '-' or op == '*' or op == '/';
}

int get_precedence(char op) {
	if (op == '+' or op == '-') return 1;
	else if (op == '*' or op == '/') return 2;
	return 0;
}

int main() {
	string expression = "3 + 4 * 2 / 1 - 5";

	stack<char> operators;
	queue<string> output;

	stringstream sin(expression);
	string token;
	while (sin >> token) {
		if (is_op(token[0])) {
			char op = token[0];
			while (!operators.empty() and get_precedence(operators.top()) >=
get_precedence(op)) {
				output.push(string(1, operators.top()));
				operators.pop();
			}
			operators.push(op);
		}
		else {
			output.push(token);
		}
	}

	// Flush operators stack
	while (!operators.empty()) {
		output.push(string(1, operators.top()));
		operators.pop();
	}
```

```
	while (!output.empty()) {
		cout << output.front() << " ";
		output.pop();
	}

	cout << endl;
}
```

## Zadatak 11

Wavefront **.obj** datoteka je obična tekstualna datoteka koja se često koristi za prikaz 3D geometrije u obliku vrhova i ploha. U ovom primjeru, svaki redak u datoteci može predstavljati vrh, trokut ili komentar. Evo primjera:

```
v 0 0 0
v 1 0 0
# This is a comment that should be ignored
v 1 1 0
v 0 1 0
f 1 2 3
f 1 3 4
```

Ako redak počinje s v, sljedeće vrijednosti su koordinate vrha. Ako redak počinje s f, sljedeće vrijednosti su (1-bazirani) indeksi vrhova koji tvore trokut. Gornji primjer prvo definira četiri vrha, a zatim dva trokuta.

Napišite program koji parsira datoteku **model.obj**, pohranjuje vrhove i trokute pomoću odgovarajuće strukture podataka te implementirati algoritam za izračun **AABB (Axis-Aligned Bounding Box)** modela — najmanjeg pravokutnog okvira koji u potpunosti sadrži objekt, poravnatog s koordinatnim osima.

Za izračun AABB-a potrebno je pronaći minimalne i maksimalne koordinate duž svake osi (x, y, z) od svih vrhova u modelu. Rezultat su dvije točke: minimalna (*x_min, y_min, z_min*) i maksimalna (*x_max, y_max, z_max*), koje zajedno definiraju pravokutni okvir koji u potpunosti obuhvaća objekt.

Program treba ispisati broj vrhova i ploha te minimalne i maksimalne koordinate okvira.

*Moguće rješenje:*

Vec3.h

```
#pragma once

struct Vec3 {
    Vec3(double x, double y, double z)
        : x(x), y(y), z(z) {
    }
    double x, y, z;
};
```

Face.h

```
#pragma once

struct Face {
    Face(int v1, int v2, int v3)
        : v1(v1), v2(v2), v3(v3) {
    }
    int v1, v2, v3;
};
```

Source.cpp

```
#include <fstream>
#include <iostream>
#include <string>
```

```cpp
#include <sstream>
#include <vector>
#include "Vec3.h"
#include "Face.h"
using namespace std;

int main() {
    ifstream fin("C:\\Users\\pero\\model.obj");
    if (!fin) {
        cout << "cannot open obj file" << endl;
        return -1;
    }

    vector<Vec3> vertices;
    vector<Face> faces;

    string line;
    float x, y, z;
    int v1, v2, v3;
    while (getline(fin, line)) {
        stringstream ss(line);
        string type;
        ss >> type;

        if (type == "v") {
            ss >> x >> y >> z;
            vertices.emplace_back(x, y, z);
        }
        else if (type == "f") {
            ss >> v1 >> v2 >> v3;
            faces.emplace_back(v1, v2, v3);
        }
    }
    fin.close();

    Vec3 min = vertices[0];
    Vec3 max = vertices[0];
    for (const Vec3& v : vertices)
    {
        if (v.x < min.x) min.x = v.x;
        if (v.y < min.y) min.y = v.y;
        if (v.z < min.z) min.z = v.z;

        if (v.x > max.x) max.x = v.x;
        if (v.y > max.y) max.y = v.y;
        if (v.z > max.z) max.z = v.z;
    }

    cout << "Vertices: " << vertices.size() << endl;
    cout << "Faces: " << faces.size() << endl;

    cout << "Bounding box:\n";
    cout << "\tMin: (" << min.x << ", " << min.y << ", " << min.z << ")\n";
    cout << "\tMax: (" << max.x << ", " << max.y << ", " << max.z << ")\n";
}
```

## Zadatak 12

Implementirajte sustav povijesti preglednika pomoću stoga. Omogućite kretanje unatrag i unaprijed, gdje korisnik može posjećivati stranice, vraćati se na prethodno posjećene stranice i ići naprijed nakon povratka. Ako nema stranica za povratak, ispišite about:blank.

Primjer rada programa:

```
> (v - visit, b - back, f – forward, q - quit): v
url: google.com
> (v - visit, b - back, f - forward, q - quit): v
url: reddit.com
> (v - visit, b - back, f - forward, q - quit): v
url: facebook.com
> (v - visit, b - back, f - forward, q - quit): f
no pages to go forward
> (v - visit, b - back, f - forward, q - quit): b
reddit.com
> (v - visit, b - back, f - forward, q - quit): f
facebook.com
> (v - visit, b - back, f - forward, q - quit): b
reddit.com
> (v - visit, b - back, f - forward, q - quit): b
google.com
> (v - visit, b - back, f - forward, q - quit): b
about:blank
> (v - visit, b - back, f - forward, q - quit): q
```

*Moguće rješenje:*

```cpp
#include <iostream>
#include <stack>
#include <string>
using namespace std;

int main() {
	stack<string> forward;
	stack<string> backward;

	while (true) {
		cout << "> (v – visit, b – back, f – forward, q – quit) ";
		if (!backward.empty())
			cout << "[" << backward.top() << "] ";
		else
			cout << "[about:blank] ";

		char op;
		cin >> op;
		cin.ignore();

		if (op == 'v') {
			cout << "url: ";
			string url;
			getline(cin, url);

			backward.push(url);
		}
		else if (op == 'b') {
```

```
                        if (backward.empty()) {
                                cout << "no pages to go back to" << endl;
                                continue;
                        }

                        forward.push(backward.top());
                        backward.pop();
                        if (!backward.empty())
                                cout << backward.top() << endl;
                }
                else if (op == 'f') {
                        if (forward.empty()) {
                                cout << "no pages to go forward" << endl;
                                continue;
                        }

                        backward.push(forward.top());
                        forward.pop();
                        if (!forward.empty())
                                cout << forward.top() << endl;
                }
                else if (op == 'q') break;
                else {
                        cout << "unknown command" << endl;
                }
        }
}
```

## Zadatak 13*

(Opcionalni) Implementirajte jednostavni interpreter Assembly jezika koji čita izvornu **.asm** datoteku. Implementacija mora podržavati sljedeći skup instrukcija (slično 6502 procesoru, s manjim izmjenama):

| | |
|---|---|
| **LDA <value \| addr>** | Loads value into register A. |
| **LDY <value \| addr>** | Loads value into register Y |
| **STA <addr>** | Store value from register A into the specified address. |
| **STY <offset>** | Store value from register A in address specified by offset + value in register Y. |
| **TAY** | Transfer value of register A to register Y. |
| **TYA** | Transfer value of register Y to register A. |
| **ADC <value \| addr>** | Adds given value to register A. |
| **INY** | Increments register Y. |
| **DEY** | Decrements register Y. |
| **JMP <line>** | Jump to specific line in the code, like GOTO directive (line numbers start with 1). |
| **JZ <line>** | Jump to a specific line in the code if value of register A is equal to 0. |

U ovoj implementaciji, memorija je raspoređena u jednostavnom cjelobrojnom nizu. Ako želite proslijediti doslovnu decimalnu vrijednost za instrukciju, morate napisati '$' ispred vrijednosti, kao u sljedećem primjeru

**LDA #$12**

što se prevodi kao učitavanje vrijednosti '12' u registar A. S druge strane, ako želite učitati vrijednost u registar A s adrese '12', koristili biste sljedeću sintaksu:

**LDA $12**

Jedna specifična naredba je STY koja pohranjuje vrijednost iz registra A na adresu određenu zadanim pomakom + vrijednošću iz registra Y. Na primjer, ako je vrijednost registra Y 14, sljedeća naredba:

**STY #$10**

pohranit će vrijednost registra A na adresu 24 (pomak = 10, Y = 14).

Pretpostavimo da izvorna datoteka sadrži valjani asemblerski kod. Odaberite odgovarajuću strukturu podataka za pohranjivanje instrukcija budući da ćete morati skakati na određene brojeve redaka. Sve literalne vrijednosti treba tretirati kao decimalne vrijednosti.

*Moguće rješenje:*

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>
using namespace std;

const int MEMORY_SIZE = 256;
```

```
int memory[MEMORY_SIZE] = { 0 };
int regA = 0, regY = 0;

int get_value(string arg) {
	if (arg[0] == '#') {
		int idx = arg[1];
		stringstream sin;
		for (int i = 2; i < arg.size(); i++)
			sin << arg[i];
		return stoi(sin.str());
	}
	stringstream sin;
	for (int i = 1; i < arg.size(); i++) {
		sin << arg[i];
	}

	return memory[stoi(sin.str())];
}

int main() {
	ifstream fin("source.asm");
	if (!fin) {
		cout << "error while opening source file\n" << endl;
	}

	vector<string> statements;
	string statement;
	while (getline(fin, statement)) {
		statements.push_back(statement);
	}
	fin.close();

	for (int i = 0; i < statements.size();)
	{
		bool jumped = false;
		stringstream sstr(statements[i]);
		string instruction, arg;
		sstr >> instruction >> arg;

		cout << instruction << endl;

		if (instruction == "LDA") {
			regA = get_value(arg);
		}
		else if (instruction == "LDY") {
			regY = get_value(arg);
		}
		else if (instruction == "STA") {
			memory[get_value(arg)] = regA;
		}
		else if (instruction == "STY") {
			int addr = get_value(arg) + regY;
			if (addr >= MEMORY_SIZE) continue;

			memory[addr] = regA;

			cout << "storing " << regA << " on " << addr << endl;
```

```
		}
		else if (instruction == "TAY") {
			regY = regA;
		}
		else if (instruction == "TYA") {
			regA = regY;
		}
		else if (instruction == "ADC") {
			regA += get_value(arg);
		}
		else if (instruction == "INY") {
			regY++;
		}
		else if (instruction == "DEY") {
			regY--;
		}
		else if (instruction == "JMP") {
			int line = stoi(arg);
			i = line - 1;
			jumped = true;
		}
		else if (instruction == "JZ") {
			if (regA == 0) {
				int line = stoi(arg);
				i = line - 1;
				jumped = true;
			}
		}

		if (!jumped) i++;
	}

	cout << "MEMORY\n";
	for (int i = 0; i < MEMORY_SIZE; i++)
		cout << "#" << i << " - " << memory[i] << endl;

	return 0;
}
```