

Skripta za samostalno učenje

UVOD U PROGRAMIRANJE C++

salabahter.eu

Varaždin, 2025.

1. Upute

Skripta je namjenjena za učenje osnovna C++ popraćena za predmet Programiranje 1 na Fakultetu Organizacije i Informatike

Obrađeno gradivo ima više kategorija težine i na zadacima i teoriji u dokumentu i zadacima biti će označeno. (Označeno je samo na stranici za rješavanje zadatka)

1=*	Lako
2=*	Srednje lako
3=**	Srednje
4=**	Teško
5=***	Vrlo teško
6=****	Napredno

Čitanjem ovog dokumenta obećajete da nećete zloupotrebljavati ove resurse, hvala.

2. Uvod u programiranje

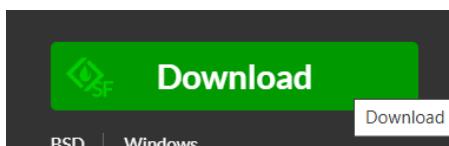
2.1. *Postavljanje sučelja za programiranje DevC++

Ovo su upute za instalaciju programa Dev-C++ na računalu sa Windows 11:

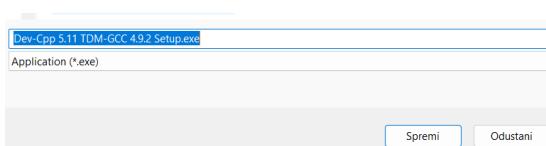
1. Preuzimanje instalacijske datoteke

*Preuzmite instalacijsku datoteku s linka: <https://sourceforge.net/projects/orwelldevcpp/>

Pritisnite tipku Download (slika 1) i preuzimanje bi trebalo početi automatski te će se otvoriti prozor s upitom (slika 2) pritisnite Spremi



Slika 1



Slika 2

2. Pokretanje instalacijske datoteke

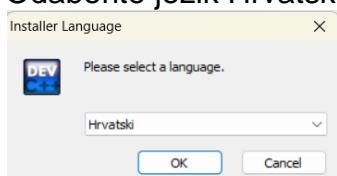
*Nakon preuzimanja, pokrenite datoteku pod nazivom:
Dev-Cpp 5.11 TDM-GCC 4.9.2 Setup.exe

Kada Vas sustav zatraži odobrenje za instalaciju, pritisnite Da.

3. Instalacija programa

Kada se instalacijski čarobnjak pokrene:

1. Odaberite jezik Hrvatski i pritisnite OK (Slika 3)



Slika 3

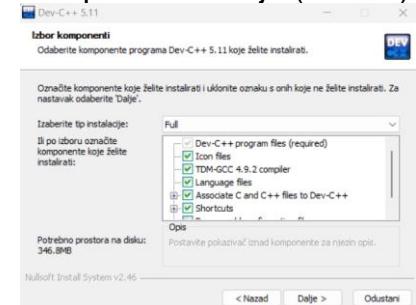
2. Pritisnite Prihvaćam (Slika 4) kako biste prihvatili uvjete licence



Slika 4

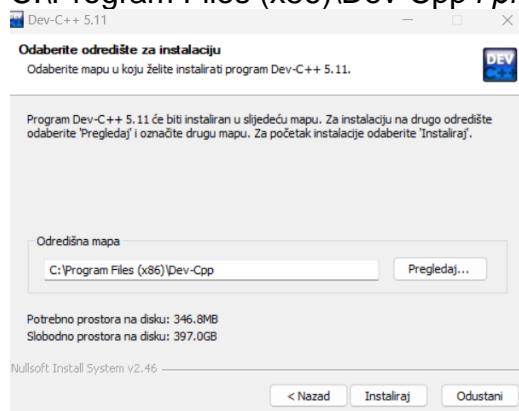
..nastavak na sljedećoj stranici..

3. Zatim pritisnite Dalje (Slika 5)



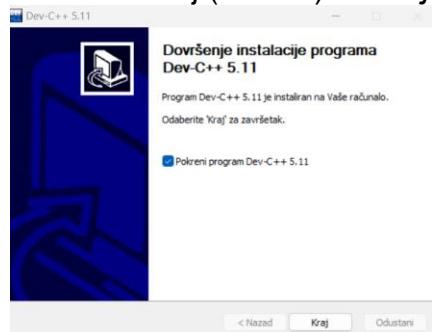
Slika 5

4. Pritisnite Instaliraj (Slika 6) *peporučuje se zadržati zadanu lokaciju za instalaciju: C:\Program Files (x86)\Dev-Cpp i pričekajte da se instalira*



Slika 6

5. Pritisnite Kraj (Slika 7) Ako nije označite da se pokrene program Dev-C++



Slika 7

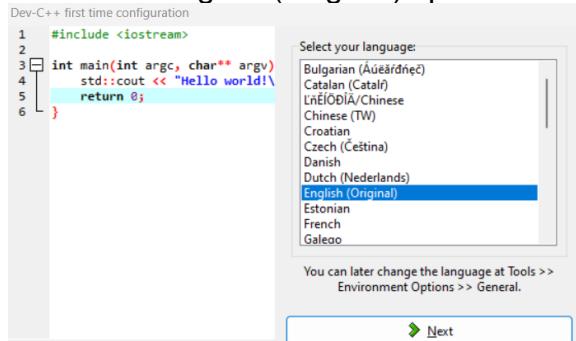
..nastavak na sljedećoj stranici..

4. Završetak instalacije i pokretanje programa

Po završetku instalacije:

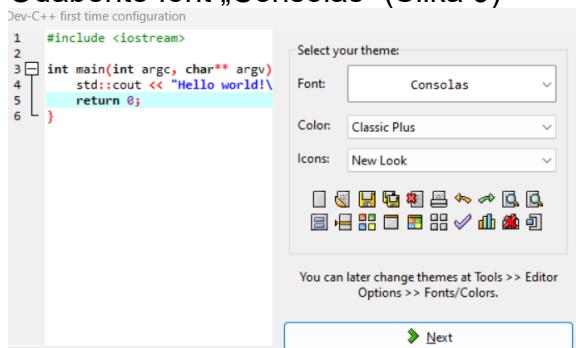
Ako se nije samostalno pokrenuo tada pokrenite program Dev-C++:

1. Odaberite English (Original) i pritisnite Next (Slika 8)



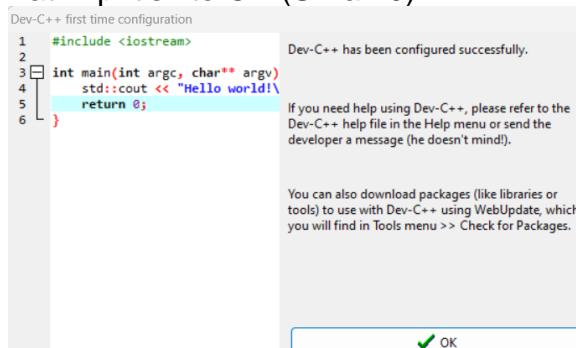
Slika 8

2. Odaberite font „Consolas“ (Slika 9)



Slika 9

3. Zatim pritisnite OK (Slika 10)



Slika 10

5. Prikaz razvojne okoline

Otvoriti će se prozor razvojne okoline nakon uspješno obavljenih koraka (Slika 11)



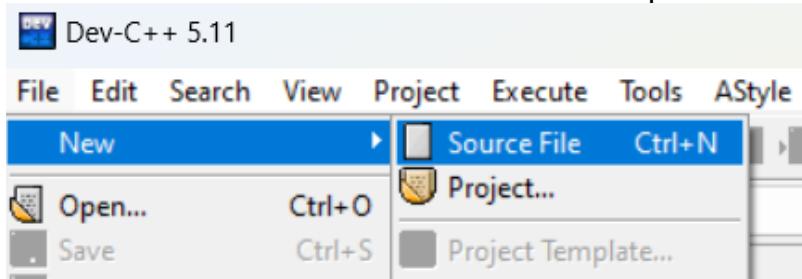
Slika 11

..nastavak na sljedećoj stranici..

6. Kreiranje nove datoteke

Za stvaranje nove prazne datoteke:

Kliknite File -> New -> Source File ili koristite prečac Ctrl+N

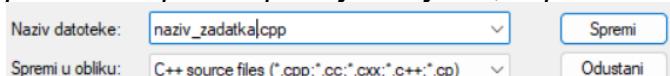


Slika 12

7. Spremanje datoteke

Za spremanje datoteke s kodom:

Kliknite File -> Save ili koristite prečac Ctrl+S *tada će vas pitati za naziv datoteke, nastojite se pridržavati pravila pisanja varijabla, i spremati datoteku u Dokumenti*

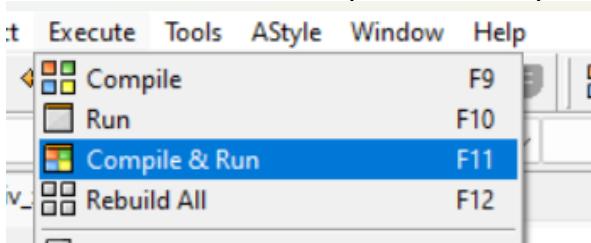


Slika 13

8. Pokretanje programa

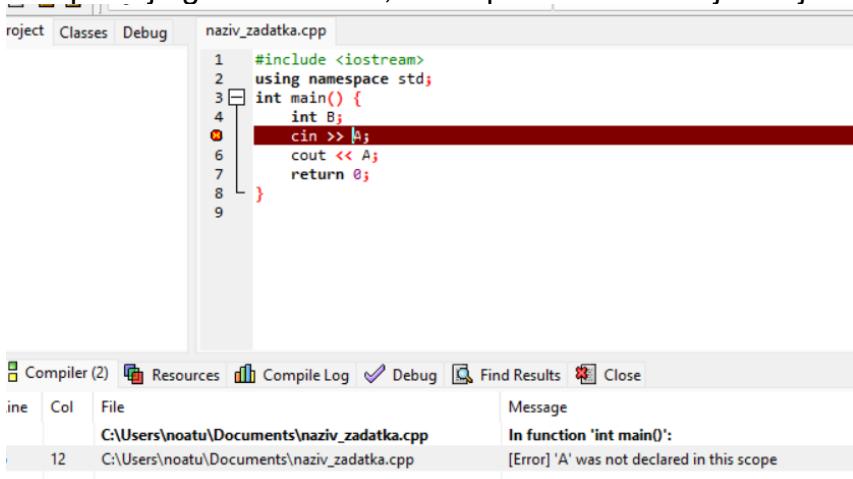
Za pokretanje napisanog C++ koda (datoteka koda mora biti spremljena korak 7.):

Kliknite Execute -> Compile & Run ili pritisnite F11.



Slika 14

Ako postoje greške u kodu, bit će prikazane u donjem dijelu prozora (Slika 15)



Slika 15

Ako je kod ispravan, program će se pokrenuti u novom terminalskom prozoru.

9. Dodatne postavke za lakše programiranje

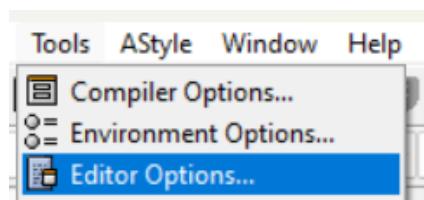
Većina osnovnih programskih kodova u c++ imaju slijedeću osnovnu strukturu:

```
#include<iostream>
using namespace std;
int main(){
    system("pause");
    return 0;
}
```

Primjer 01

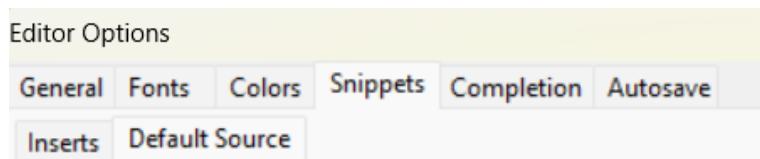
Kako bi se olakšalo programiranje, složite da svaki novi program ima tu strukturu već na početku.

1. Kako biste to postigli pritisnite Tools->Editor Options



Slika 16

2. Zatim odaberite karticu Snippets->Default Source

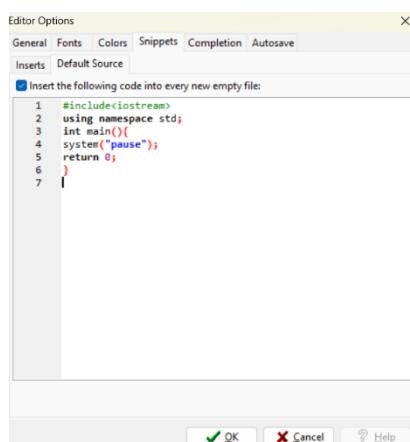


Slika 17

3. Zatim kod *Primjer 01* umetnите u prazni prostor i označite checkbox te pritisnite OK

```
#include<iostream>
using namespace std;
int main(){
    system("pause");
    return 0;
}
```

Primjer 01



Slika 18

2.2. *Objašnjenje osnovnih pojmoveva u programu

C++ program sastoji se od:

- **Deklaracijskog dijela:** Koristi se za deklaraciju varijabli, funkcija i uključivanje knjižnica potrebnih za rad programa.
- **Izvedbenog dijela:** Glavno tijelo programa koje se nalazi unutar funkcije main() i gdje se naredbe izvršavaju sekvensijalno.



Biblioteke u C++

Biblioteke su skupovi prethodno napisanih funkcija i klasa koje programeri mogu koristiti za ubrzanje razvoja programa. Omogućuju pristup osnovnim funkcijama poput ulazno-izlaznih operacija, matematičkih izračuna i manipulacije stringovima.

Standardna knjižnica <iostream>: Služi za rad s ulazom i izlazom podataka i osnovnom programerskom logikom. Omogućuje korištenje cin za unos i cout za ispis podataka. Uključivanje knjižnice: #include <iostream>

Ključna riječ **using namespace** omogućuje upotrebu elemenata iz zadanog prostora imena bez potrebe za navođenjem imena prostora svaki put.

Primjer bez using namespace:

```
#include <iostream>
int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Primjer s using namespace std:

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

using namespace std izbjegava potrebu za std:: ispred cin, cout i endl

cin - unos podataka

cin omogućuje unos podataka s tipkovnice. Koristi se za učitavanje vrijednosti u varijable:

```
cin >> i; // Moguće unositi više varijabli odjednom: cin >> i >> j >> k;
```

cout - ispis podataka i endl (novi red)

cout omogućuje ispis podataka na standardni izlaz (obično zaslon):

```
cout << i << endl; // `endl` se koristi za novi red
```

Komentari se koriste za objašnjavanje koda i ignoriraju se prilikom izvršavanja programa.

```
// Ovo je jednoredni komentar
int x = 5; // Varijabla x dobiva vrijednost 5
```

```
/* Ovo je višeredni komentar koji se proteže
preko više linija koda.*/
```

Aritmetički operatori

Operator	Značenje	Primjer
+	Zbrajanje	A + B
-	Oduzimanje	A - B
*	Množenje	A * B
/	Dijeljenje	A / B
%	Modulo (ostatak)	A % B

Složeni operatori dodjele

```
S = S + A; // Ekvivalentno: S += A
```

```
S = S - A; // Ekvivalentno: S -= A
```

```
S = S * A; // Ekvivalentno: S *= A
```

```
S = S / A; // Ekvivalentno: S /= A
```

```
S = S % A; // Ekvivalentno: S %= A
```

```
B += 1; // B++ povećava vrijednost nakon korištenja, ++B prije korištenja
```

```
B -= 1; // B-- smanjuje vrijednost nakon korištenja, --B prije korištenja
```

3. Varijable i tipovi podataka

3.1. Teorija

Varijabla je imenovani dio memorije koji sadrži podatak korišten u programu. Vrijednost varijable se tijekom izvršavanja programa može mijenjati.

ELI5 Objasnjenje:

Zamislite varijablu kao kutiju s etiketom koja kaže što se u njoj može nalaziti. Ako je kutija označena s "brojevi", unutra mogu biti samo brojevi. Ako je označena s "slova", unutra mogu biti samo slova. Ne možete staviti slovo u kutiju za brojeve niti broj u kutiju za slova. Varijabla omogućuje da spremimo podatke i promijenimo ih, ali tip podataka mora uvijek ostati isti.

Deklaracija i inicijalizacija

Da bi se varijabla u programu mogla koristiti varijabla se mora deklarirati. **Deklariranje** varijable u C++-u znači rezerviranje memoriskog prostora za pohranu podataka određenog tipa. Prilikom deklaracije određujemo ime varijable i tip podataka koji će ona sadržavati, ali još joj ne dodjeljujemo konkretnu vrijednost u sljedećem obliku:

tip_podatka Naziv_Varijable

Inicijaliziranje varijable znači dodjeljivanje početne vrijednosti varijabli u trenutku deklaracije ili kasnije. Inicijalizacija osigurava da varijabla ima definiranu vrijednost prilikom korištenja.

Oba primjera su ekvivalenta:

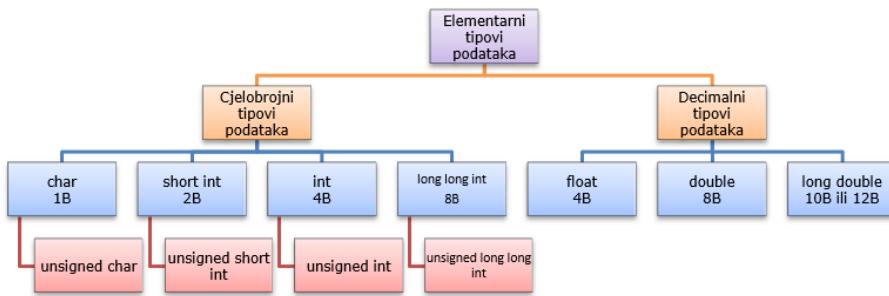
```
int broj; //deklaracija varijable  
broj = 5; //inicijalizacija varijable  
int broj = 5; // deklaracija i inicijalizacija u jednom koraku
```

Razlika:

- **Deklaracija:** Samo najava varijable i njenog tipa.
- **Inicijalizacija:** Postavljanje početne vrijednosti varijabli.

Tipovi podataka

Tip podataka određuje vrstu vrijednosti koju varijabla može pohraniti i količinu memorije koja će biti zauzeta.



Slika 19

Što je bajt [B]?

Bajt je osnovna jedinica za pohranu podataka u računalima i sastoji se od 8 bitova. Svaki bit može biti 0 ili 1, što omogućuje pohranu binarnih podataka. Na primjer, 1 bajt može pohraniti cijeli broj od -128 do 127 tako da svaki bit predstavlja određenu težinu.

Cjelobrojni tipovi:

- **char:** 1 bajt, raspon vrijednosti od -128 do 127. Koristi se za pohranu znakova i manjih cijelih brojeva.
- **short int:** 2 bajta, raspon od -32,768 do 32,767. Koristi se za pohranu manjih cijelih brojeva.
- **int:** 4 bajta, raspon od -2,147,483,648 do 2,147,483,647. Najčešće korišteni tip za pohranu cijelih brojeva.
- **long long int:** 8 bajta, raspon od približno -9×10^{18} do 9×10^{18} . Koristi se za pohranu vrlo velikih cijelih brojeva.

Zašto različiti tipovi?

Tipovi se razlikuju kako bi omogućili optimizaciju memorije. Manji tipovi koriste manje memorije, dok veći tipovi omogućuju pohranu većih brojeva.

Decimalni tipovi:

- **float:** 4 bajta, točnost 7 decimala - za pohranu brojeva s pomičnim zarezom
- **double:** 8 bajta, točnost 15 decimala. za rad s decimalnim brojevima
- **long double:** veličina ovisi o prevoditelju, nudi veću preciznost od double

Zašto koristiti različite decimalne tipove? Float koristi manje memorije, ali ima manju preciznost u usporedbi s double i long double. Za znanstvene izračune koristi se double ili long double radi preciznosti.

Unsigned (bez predznaka): Koristi se kada znamo da vrijednost nikada neće biti negativna, ODNOSNO DA ĆE BITI POZITIVNA CIJELOBROJNA VARIJABLA (kod zadatka obratiti pažnju na to).

3.2. Primjeri zadataka

ID	74
Zadatak	Deklarirajte varijablu tipa int pod imenom x i dodijelite joj vrijednost 10. Zatim ispišite tu vrijednost na ekran.
Riješenje	<pre>#include <iostream> using namespace std; int main() { int x = 10; cout << "Vrijednost broja: " << x << endl; return 0; }</pre>
Objašnj.	int x = 10; //deklaracija i incijalizacija varijable x na cijeli broj 10 cout << "Vrijednost broja: " << x << endl; //ispis teksta i varijable x
Ulaz	
Izlaz	Vrijednost broja: 10
Težina	1-Lako

ID	75
Zadatak	Deklarirajte varijablu tipa float pod imenom decimalniBroj. Korisnik unosi vrijednost, a program je ispisuje na ekran.
Riješenje	<pre>#include <iostream> using namespace std; int main() { float decimalniBroj; cout << "Unesite decimalni broj: "; cin >> decimalniBroj; cout << decimalniBroj << endl; return 0; }</pre>
Objašnj.	cin >> decimalniBroj; //unos decimalnog broj u varijablu decimalniBroj
Ulaz	5.5
Izlaz	5.5
Težina	1-Lako

ID	76
Zadatak	Deklarirajte pozitivna cijelobrojna broja a i b. Unesite njihove vrijednosti i ispišite zbroj, razliku, umnožak i kvocijent.
Riješenje	<pre>#include <iostream> using namespace std; int main() { unsigned int a, b; //unsigned jer su pozitivni cin >> a >> b; cout << a + b << endl; //zbroj cout << a - b << endl; //razlika cout << a * b << endl; //umnožak cout << a / b << endl; //kvocijent return 0; }</pre>
Objašnj.	cout << "Kvocijent: " << a / b << endl; //da nije u tekstu zadatka navedeno da su pozitivni cijeli brojevi, nego da piše samo cijeli brojevi došlo bi do greške kada bi varijabla a bila jednaka 0.
Ulaz	5 10
Izlaz	15 -5 50 0
Težina	2-Srednje lako

ID	77
Zadatak	Unesite cijeli broj x. Ispišite vrijednost x+1, x-1 i x*2 svaki u novi red.
Riješenje	<pre>#include <iostream> using namespace std; int main() { int x; cin >> x; cout << x+1 << endl << x-1 << endl << x*2; return 0; }</pre>
Objašnj.	
Ulaz	10
Izlaz	11 9 20
Težina	2-Srednje lako

ID	78
Zadatak	Napišite program u kojem korisnik unosi decimalni broj x. Program prvo treba ispisati cijelobrojni dio varijable. U novom retku treba ispisati decimalni dio varijable.
Riješenje	<pre>#include <iostream> using namespace std; int main() { double x; cin >> x; cout <<(int)x<<endl<<x-(int)x; return 0; }</pre>
Objašnj.	cout <<(int)x<<endl<<x-(int)x; // zagrada ispred naziva varijable služi za pretvorbu u drugi tip podataka
Ulaz	56.57
Izlaz	56 0.57
Težina	4-Srednje

ID	79
Zadatak	Napišite program u kojem korisnik unosi 3 cijelih pozitivnih brojeva i jedan decimalni broj. Program ispisuje sve brojeve u istom redu, obrnutim redoslijedu. Nakon toga u novom redu ispisuje sumu svih unesenih brojeva. (riješiti bez korištenja petlja)
Riješenje	<pre>#include <iostream> using namespace std; int main() { unsigned int a, b, c; //deklariranje pozitivnih cijelih brojeva float d; //deklariranje decimalnog broja cin >> a >> b >> c >> d; //unos cout << d << " " << c << " " << b << " " << a; float suma = a + b + c + d; //izračun sume tip float cout << endl << suma; //ispis sume return 0; }</pre>
Objašnj.	Varijabla suma je tipa float jer se u nju zbraja decimalni broj (d) zajedno s cijelim brojevima (a, b, c). Kada se cijeli brojevi zbrajamaju s decimalnim brojem, rezultat se automatski proširuje na tip s većom preciznošću kako bi se sačuvala decimalna točnost. U ovom slučaju, float osigurava da se decimalni dio broja ne izgubi prilikom izračuna.
Ulaz	5 10 15 3.5
Izlaz	3.5 15 10 5 33.5
Težina	4-Srednje

4. Selekcije, operatori i operacije

Selekcije omogućuju izvršavanje dijela koda ovisno o istinitosti logičkog izraza. Koriste se za donošenje odluka u programu.

```
if (A > B) {  
    cout << "A je veće od B";  
} else {  
    cout << "A nije veće od B";  
}
```

Operatori usporedbe

<, >, <=, >=, ==, !=

Selekcija tipa switch

```
switch (cjelobrojni izraz) {  
    case vrijednost1:  
        blok naredbi;  
        break;  
    case vrijednost2:  
        blok naredbi;  
        break;  
    default:  
        blok naredbi;  
}
```

Skraćeni zapis if-else strukture

```
izraz ? vrijednost1 : vrijednost2;
```

```
int a = 5, b = 10;  
  
int max = (a > b) ? a : b;
```

Ukratko

- **Aritmetički operatori:** +, -, *, /, %
- **Relacijski operatori:** <, >, <=, >=, ==, !=
- **Logički operatori:**
 - ! - negacija
 - && - konjunkcija
 - || - disjunkcija

5. Iteracije i skokovi

Iteracije omogućuju ponavljanje određenog dijela koda dok je zadani uvjet ispunjen.

Vrste iteracija:

For petlja (s eksplisitnim brojačem): Koristi se kada je poznat broj ponavljanja unaprijed.

```
for (inicijalizacija; uvjet; promjena) {  
    // tijelo petlje  
}  
  
for (int i = 1; i <= 5; i++) {  
    cout << i << " ";  
}
```

While petlja (logički uvjet na početku): Koristi se kada uvjet odlučuje o ulasku u petlju.

```
while (uvjet) {  
    // tijelo petlje  
}  
  
int i = 1;  
  
while (i <= 5) {  
    cout << i << " ";  
    i++;  
}
```

Do-While petlja (logički uvjet na kraju): Tijelo petlje se izvršava barem jednom.

```
do {  
    // tijelo petlje  
} while (uvjet);  
  
int i = 1;  
  
do {  
    cout << i << " ";  
    i++;  
} while (i <= 5);
```

PRIMJER

```
Ispis prostih brojeva do broja N
#include <iostream>

#include <cmath>

using namespace std;

int main() {
    unsigned short N;
    cout << "Upisite N: ";
    cin >> N;
    bool prost = true;
    for (unsigned short k = 2; k <= sqrt(N); k++) {
        if (N % k == 0) {
            prost = false;
            break;
        }
    }
    if (prost)
        cout << "Broj " << N << " je prost.";
    else
        cout << "Broj " << N << " nije prost.";
    return 0;
}
```

6. Polja

6.1. Teorija

Sintaksa deklaracije polja: **tip_podatka identifikator[broj_elemenata];**

Za polje: **int p[5]** ima 5 elementa a pristupa im se promjenom broja u području broj_elemenata

Prvi element svakog polja je na poziciji p[0] i zato je zadnji p[n-1] znači ako je n=5 odnosno broj elemenata jednak 5 prvi element je na poziciji p[0] a zadnji na p[4]

Važno:

- Varijable u polju su **sve istog tipa** podataka
- Smještene su na uzastopnim memorijskim lokacijama.

Rad s matricama

Zadatak:

Pronaći maksimalnu i minimalnu vrijednost u matrici M×NM \times NM te njihove pozicije.

Logika:

Generirati pseudoslučajne vrijednosti za matricu.

Iterirati kroz sve elemente matrice.

Pronaći minimum i maksimum, te njihove indekse.

```
int M, N, A[100][100];  
  
cin >> M >> N;  
  
srand(time(0));  
  
for (int i = 0; i < M; i++) {  
  
    for (int j = 0; j < N; j++) A[i][j] = rand() % 100;  
}  
  
int min = A[0][0], max = A[0][0];  
  
int minRow = 0, minCol = 0, maxRow = 0, maxCol = 0;  
  
for (int i = 0; i < M; i++) {  
  
    for (int j = 0; j < N; j++) {  
  
        if (A[i][j] < min) {  
  
            min = A[i][j];  
  
            minRow = i;  
  
            minCol = j;  
        }  
  
        if (A[i][j] > max) {  
  
            max = A[i][j];  
  
            maxRow = i;  
  
            maxCol = j;  
        }  
    }  
}  
  
cout << "Min: A(" << minRow << "," << minCol << ") = " << min << endl;  
cout << "Max: A(" << maxRow << "," << maxCol << ") = " << max << endl;
```

6.2. Zadaci/primjeri

126 - PD Eratostenovo sito

POLJA

Prirodni brojevi N i M koji predstavljaju dvije granice u rasponu <1, 1000]. Svi prosti brojevi između N i M izračunati koristeći Eratostenovo sito.

Preuzeto s progdemos repozitorija

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    int N, M;
    // Korisnički unos donje granice N
    do {
        cout << "Unesite donju granicu N <1, 1000>: ";
        cin >> N;
    } while (N < 1 || N >= 1000); // Provjera da li je N unutar dozvoljenog
//raspona
    // Korisnički unos gornje granice M
    do {
        cout << "Unesite gornju granicu M <" << N << ", 1000>: ";
        cin >> M;
    } while (M < N || M > 1000); // Provjera da li je M unutar dozvoljenog
//raspona i veći ili jednak N

    // Polje za označavanje prostih brojeva
    bool prost[1001]; // Polje za sve brojeve do 1000 (indeksi 0-1000)

    // Inicijalno postavljamo sve brojeve od 2 do M kao potencijalno proste
    for (int i = 2; i <= M; i++) prost[i] = true;

    // Implementacija Eratostenovog sita
    for (int i = 2; i <= sqrt(M); i++) { // Iteriramo do korijena od M
        if (prost[i]) { // Ako je broj i označen kao prost
            for (int j = i * 2; j <= M; j += i) { // Označavamo sve
//višekratnike broja i kao neproste
                prost[j] = false;
            }
        }
    }

    // Ispis prostih brojeva unutar granica [N, M]
    for (int i = N; i <= M; i++) {
        if (prost[i]) // Ako je broj i označen kao prost
            cout << i << endl; // Ispis prostog broja
    }

    return 0;
}
```

Provjeriti je li riječ S palindrom (rijec koja se čita spreda i straga jednako).

```
#include <iostream>
#include <cstring>
using namespace std;
int main () {
    char S[20];
    bool pal = true;
    cout << "S = ";
    cin >> S;
    int L = strlen(S);
    for (int i=0; i<L/2 && pal; i++)
        if (S[i] != S[L-i-1])
            pal = false;
    if (pal)
        cout << "Rijec je palindrom"
            << endl;
    else
        cout << "Rijec nije palindrom"
            << endl;
    return 0;
}
```

7. Pretraživanje i sortiranje

7.1. Teorija

Sortiranje je proces reorganizacije elemenata u nizu tako da su uređeni prema određenom kriteriju (npr. rastući ili padajući redoslijed).

7.2. Primjeri**Pretraživanje

Slijedno pretraživanje polja

```
#include <iostream>
using namespace std;
int main () {
    int N;
    float A[1000], K;

    do {
        cout << "N = ";
        cin >> N;
    } while (N < 1 || N > 1000);

    for (int i = 0; i < N; i++) {
        cout << "A[" << i << "] = ";
        cin >> A[i];
    }

    cout << "K = ";
    cin >> K;

    int i = 0;
    bool nadjen = false;
    while (!nadjen && i < N) {
        if (A[i] == K) nadjen = true;
        else i++;
    }

    if (nadjen) cout << i;
    else cout << -1;
}

return 0;
}
```

7.3. Primjeri-Osnovna sortiranja



Svaki primjer objašnjen je na poveznici:

<https://chatgpt.com/share/677c18b8-167c-8008-8b6b-30228913bc96>

Sortiranje izborom**

```
for (int i = N - 1; i > 0; i--) {  
    int Max = 0;  
    for (int j = 1; j <= i; j++) {  
        if (A[j] > A[Max]) {  
            Max = j;  
        }  
    }  
    float pom = A[i];  
    A[i] = A[Max];  
    A[Max] = pom;  
}
```

Sortiranje zamjenom*

```
for (int i = N-1; i > 0; i--) {  
    for (int j = 0; j < i; j++)  
        if (A[j] > A[i]) {  
            float pom = A[j];  
            A[j] = A[i];  
            A[i] = pom;  
        }  
}
```

Mjehuričasto sortiranje**

```
bool zamjena = true;  
for (int i = N-1; i > 0 && zamjena; i--) {  
    zamjena = false;  
    for (int j = 0; j < i; j++)  
        if (A[j] > A[j+1]) {  
            float pom = A[j];  
            A[j] = A[j+1];  
            A[j+1] = pom;  
            zamjena = true;  
        }  
}
```

Sortiranje umetanjem***

```
for (int i = 1; i < N; i++) {  
    int j = i-1;  
    float pom = A[i];  
    while (j >= 0 && A[j] > pom){  
        A[j+1] = A[j--];}  
    A[j+1] = pom;  
}
```

7.4. Primjeri-Sortiranje dodatno

Rbr.	73a
Zadtk.	Korisnik unosi N broj riječi koji se tada sortiraju po abecednom redoslijedu
Rj.	<pre>#include <iostream> #include <cstring> using namespace std; int main() { int N; char A[1000][50]; // Unos broja rijeci do { cin >> N; } while (N < 2 N > 1000); // Unos rijeci for (int i = 0; i < N; ++i) { cin >> A[i]; } // Sortiranje rijeci koristeci sortiranje umetanjem for (int i = 1; i < N; ++i) { char temp[50]; strcpy(temp, A[i]); int j = i - 1; while (j >= 0 && strcmp(A[j], temp) > 0) { strcpy(A[j + 1], A[j]); --j; } strcpy(A[j + 1], temp); } // Ispis sortiranih rijeci for (int i = 0; i < N; ++i) cout << A[i] << " "; cout << endl; return 0; }</pre>
Objš.	strcpy(destinacija, izvor);
Ulaz	3 Gorko Ana Bicikl
Izlaz	Ana Bicikl Gorko
Težina	4- Teško

8. Slogovi i unije

8.1. Teorija

Slog je mehanizam za agregaciju podataka različitih tipova pod jednim imenom.
Svaki podatak unutar sloga naziva se komponenta.

```
struct ImeSloga {
    tip_podataka komponenta1;
    tip_podataka komponenta2;
    ...
};

struct ClanKnjiznice {
    char ime[20];
    char prezime[20];
    int brojOsobneIskaznice;
};
```

Korištenje operatora za dohvatanje članova (.) ili pokazivača (->).

Unija omogućuje pohranjivanje više komponenata na istoj memorijskoj lokaciji.
Samo jedna komponenta unije može sadržavati vrijednost u bilo kojem trenutku.

```
union ImeUnije {
    tip_podataka komponenta1;
    tip_podataka komponenta2;
    ...
};

union PrimjerUnije {
    char znak;
    int cijelaVrijednost;
    float decimalnaVrijednost;
};
```

Koristi se kada imamo međusobno isključive podatke i želimo efikasnije koristiti memoriju.

Veličina unije je veličina njezinog najvećeg tipa podataka.

Popunjavanje sa paddingom na primjeru:

- 1) char X zauzima 1 bajt, ali sljedeći član (long long int) zahtijeva poravnjanje na 8 bajtova.
- 2) Zato se dodaje 7 bajtova paddinga kako bi Y započeo na adresnom višekratniku 8.
- 3) Y počinje na višekratniku **8** i zauzima **8 bajtova**.
- 4) Sljedeći član (int) zahtijeva poravnjanje na **4 bajta**, što je već zadovoljeno.
- 5) Z zauzima **4 bajta**.
- 6) Da bi cijela struktura bila **poravnata na najveći tip** (8 bajtova zbog long long int), dodaje se **4 bajta paddinga**.

X	p	p	p	p	p	p	p
Y	Y	Y	Y	Y	Y	Y	Y
Z	Z	Z	Z	p	p	p	p

```
Struct Struktura{
    char X; //zauzima 1B
    long long int Y; //zauzima 8B
    int Z; //zauzima 4B
}
```

Enumeracija (Enumeration)

Enumeracija je način grupiranja cjelobrojnih konstanti u programski definiran tip podatka.

```
enum ImeEnumeracije {
    vrijednost1,
    vrijednost2,
    ...
};

enum DaniUTjednu {PON, UTO, SRI, ČET, PET, SUB, NED};
DaniUTjednu danas = PON;
```

Vrste enumeracija:

- Dosežujuća (enum struct): Komponente nisu dostupne izvan dosega.
- Nedosežujuća (enum): Komponente su dostupne unutar dosega.

8.2. Primjeri

Primjer: Slog za člana knjižnice

```
#include <iostream>
#include <cstring> // za strcpy
using namespace std;

// Definicija sloga
struct ClanKnjiznice {
    char ime[30];
    char prezime[30];
    int brojOsobneIskaznice;
    int brojPosudenihKnjiga;
};

int main() {
    // Deklaracija varijable sloga
    ClanKnjiznice clan;

    // Inicijalizacija komponenata
    strcpy(clan.ime, "Ivan");
    strcpy(clan.prezime, "Horvat");
    clan.brojOsobneIskaznice = 123456789;
    clan.brojPosudenihKnjiga = 5;

    // Pritup komponentama sloga
    cout << "Ime: " << clan.ime << endl;
    cout << "Prezime: " << clan.prezime << endl;
    cout << "Broj osobne iskaznice: " << clan.brojOsobneIskaznice << endl;
    cout << "Broj posuđenih knjiga: " << clan.brojPosudenihKnjiga << endl;

    return 0;
}
```

```

Slog za točku u 2D prostoru **obrađeno kasnije**
#include <iostream>
using namespace std;

// Definicija sloga
struct Tocka {
    int x;
    int y;
};

int main() {
    // Deklaracija varijable i pokazivača
    Tocka t1 = {5, 10};
    Tocka* pt = &t1; // Pokazivač na slog

    // Pristup komponentama preko pokazivača
    cout << "Koordinate točke: (" << pt->x << ", " << pt->y << ")" << endl;

    // Modifikacija vrijednosti putem pokazivača
    pt->x = 20;
    pt->y = 30;
    cout << "Ažurirane koordinate: (" << pt->x << ", " << pt->y << ")" << endl;

    return 0;
}

```

UNIJA: Pohrana različitih tipova podataka

```

#include <iostream>
using namespace std;
// Definicija unije
union Podatak {
    int cijeliBroj;
    float decimalniBroj;
    char znak;
};

int main() {
    // Deklaracija varijable unije
    Podatak podatak;
    // Postavljanje i ispisivanje različitih vrijednosti
    podatak.cijeliBroj = 42;
    cout << "Cijeli broj: " << podatak.cijeliBroj << endl;

    podatak.decimalniBroj = 3.14f;
    cout << "Decimalni broj: " << podatak.decimalniBroj << endl;

    podatak.znak = 'A';
    cout << "Znak: " << podatak.znak << endl;

    return 0;
}

```

.

Enumeracija: Dani u tjednu

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

// Definicija enumeracije
enum DaniUTjednu {PON, UTO, SRI, CET, PET, SUB, NED};

int main() {
    // Deklaracija varijable enumeracije
    DaniUTjednu danas = PET;

    // Provjera dana
    if (danас == PET) {
        cout << "Danas je petak!" << endl;
    } else {
        cout << "Danas nije petak." << endl;
    }

    return 0;
}
```

Kombinacija sloga i unije: Studentski podaci
--

```
#include <iostream>

#include <cstring>

using namespace std;

// Definicija unije

union Kontakt {
    char email[50];
    char telefon[15];
};

// Definicija sloga

struct Student {
    char ime[30];
    Kontakt kontakt;
    bool koristiEmail; // true za email, false za telefon
}; //NASTAVAK DALJE
```

```
int main() {
    // Deklaracija studenta
    Student s;
    strcpy(s.ime, "Ana");
    // Postavljanje emaila
    s.koristiEmail = true;
    strcpy(s.kontakt.email, "ana@gmail.com");
    // Ispis podataka
    cout << "Ime studenta: " << s.ime << endl;
    if (s.koristiEmail) {
        cout << "Kontakt (Email): " << s.kontakt.email << endl;
    } else {
        cout << "Kontakt (Telefon): " << s.kontakt.telefon << endl;
    }
    return 0;
}
```

Polje slogova: Slog za zaposlenike

```
#include <iostream>
#include <cstring>
using namespace std;
// Definicija sloga
struct Zaposlenik {
    char ime[30];
    float placa;
};
```

```
int main() {
    // Deklaracija polja slogova
    Zaposlenik zaposlenici[3];

    // Inicijalizacija zaposlenika
    strcpy(zaposlenici[0].ime, "Marko");
    zaposlenici[0].placa = 1000.0f;

    strcpy(zaposlenici[1].ime, "Ana");
    zaposlenici[1].placa = 1200.0f;

    strcpy(zaposlenici[2].ime, "Ivan");
    zaposlenici[2].placa = 1100.0f;

    // Ispis podataka
    for (int i = 0; i < 3; i++) {
        cout << "Ime: " << zaposlenici[i].ime << ", Plaća: " <<
zaposlenici[i].placa << " kn" << endl;
    }

    return 0;
}
```

9. Pokazivači

. Što su pokazivači?

- **Definicija:** Pokazivač je varijabla koja na svojoj memorijskoj adresi sadrži adresu druge varijable.
- **Sintaksa:**
 - tipPodatka *imePokazivača;
 - Primjer:
 - int *pokInt;

2. Zašto koristiti pokazivače?

- Omogućuju rad s dinamički alociranom memorijom.
- Korisni su za implementaciju kompleksnih struktura podataka (npr. vezane liste).
- Smanjuju potrebu za duplicitanjem podataka.

3. Osnovni primjer

```
// predprocesorski dio
#include <iostream>
using namespace std;

// glavna funkcija
int main()
{
    int varInt = 5; // definiranje varijable
    int * pokInt = &varInt; // definiranje pokazivača
    // za dohvaćanje adrese koristimo operator adrese '&'
    // odnosno adresni operator

    // ispis
    cout << varInt << " " << pokInt << " " << *pokInt << endl;
    return 0;
}

// ispis: 5 [adresa od varInt] 5
```

```
int varInt = 5;      // definiranje varijable
int * pokInt = &varInt;  // definiranje pokazivača
// za dohvaćanje adrese koristimo operator adrese '&'

// ispis
cout << *pokInt << endl;
// 5

// pridruživanje vrijednosti
*pokInt = 10;

// ispis
cout << *pokInt << endl;
// 10
```

Primjeri

Pokazivač na pokazivač

```
#include <iostream>
using namespace std;
int main() {
    int varInt = 5;
    int* pokInt = &varInt; // Pokazivač na varInt
    int** pokPok = &pokInt; // Pokazivač na pokazivač
    // Ispis razina pokazivača
    cout << "Vrijednost varInt: " << varInt << endl;
    cout << "Vrijednost varInt preko pokazivača: " << *pokInt << endl;
    cout << "Vrijednost varInt preko pokazivača na pokazivač: " << **pokPok << endl;
    return 0;
}
```

Dinamička alokacija memorije

```
#include <iostream>
using namespace std;
int main() {
    int velicina;
    cout << "Unesite veličinu polja: ";
    cin >> velicina;
    int* dinNiz = new int[velicina]; // Pokazivač za dinamičko polje i alokacija
    // Popunjavanje polja
    for (int i = 0; i < velicina; i++) {
        dinNiz[i] = i * 2; // Dodavanje vrijednosti
    }
    // Ispis polja
    cout << "Elementi polja: ";
    for (int i = 0; i < velicina; i++) {
        cout << dinNiz[i] << " ";
    }
    cout << endl;
    delete[] dinNiz; // Oslobađanje memorije
    return 0;
}
```

Nulirani pokazivač

```
#include <iostream>
using namespace std;
int main() {
    int* pokInt = NULL; // Inicijalizacija nuliranog pokazivača
    if (!pokInt) { // Provjera je li pokazivač nuliran
        cout << "Pokazivač ne pokazuje na validan objekt." << endl;
    }
    int varInt = 10;
    pokInt = &varInt; // Postavljanje pokazivača na adresu varInt

    if (pokInt) {
        cout << "Pokazivač sada pokazuje na: " << *pokInt << endl;
    }
    return 0;
}
```

Aritmetika pokazivača

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

int main() {
    int niz[] = {2, 4, 6, 8, 10};
    int* pokInt = niz; // Pokazivač na prvi element niza

    // Ispis elemenata pomoću aritmetike pokazivača
    for (int i = 0; i < 5; i++) {
        cout << "Element " << i << ": " << *(pokInt + i) << endl;
    }

    return 0;
}
```

Pokazivač na dinamički alocirano polje

```
#include <iostream>
using namespace std;
int main() {
    int velicina = 3;
    int* polje = new int[velicina]; // Dinamička alokacija polja
    // Popunjavanje polja
    for (int i = 0; i < velicina; i++) {
        polje[i] = (i + 1) * 5;
    }
    // Ispis polja
    cout << "Elementi polja: ";
    for (int i = 0; i < velicina; i++) {
        cout << polje[i] << " ";
    }
    cout << endl;
    delete[] polje; // Oslobođanje memorije
    return 0;
}
```

Pokazivač i reference

```
#include <iostream>
using namespace std;
int main() {
    int varInt = 42;
    int& refInt = varInt; // Referenca na varInt
    int* pokInt = &varInt; // Pokazivač na varInt

    cout << "Vrijednost putem reference: " << refInt << endl;
    cout << "Vrijednost putem pokazivača: " << *pokInt << endl;

    // Promjena vrijednosti putem reference
    refInt = 100;
    cout << "Nova vrijednost: " << varInt << endl;
    return 0;
}
```

10. Funkcije

Što su funkcije?

- Funkcije su blokovi koda koji izvršavaju određeni zadatak i mogu se ponovno koristiti u programu.
- Pomažu u:
 - Smanjenju duplicitiranja koda.
 - Boljoj organizaciji i čitljivosti programa.
 - Lakšem održavanju koda.

Struktura funkcije

Svaka funkcija ima tri glavne komponente:

1. **Povratni tip:**
 - Označava tip podatka koji funkcija vraća (npr. int, float, void).
2. **Naziv funkcije:**
 - Identifikator koji se koristi za pozivanje funkcije.
3. **Parametri (opcionalno):**
 - Ulazni podaci koje funkcija koristi.
4. **Tijelo funkcije:**
 - Blok koda koji se izvršava kada se funkcija pozove

```
int Saberi(int a, int b) {  
    return a + b; // Povratna vrijednost  
}
```

Deklaracija i definicija

Deklaracija: Informira kompjajler o postojanju funkcije (obično se koristi u zaglavlju koda)

```
int Saberi(int a, int b) { //Deklaracija  
    return a + b; // Definicija  
}
```

Definicija: Sadrži stvarnu implementaciju funkcije

Primjeri

Funkcija s povratnom vrijednošću – izračun faktorijela

```
#include <iostream>  
using namespace std;  
// Funkcija za izračun faktorijela  
int Faktorijel(int broj) {  
    int rezultat = 1;  
    for (int i = 1; i <= broj; i++) {  
        rezultat *= i;  
    }  
    return rezultat;  
}  
int main() {  
    int broj = 5;  
    cout << "Faktorijel od " << broj << " je " << Faktorijel(broj) << endl;  
    return 0;  
}
```

Preopterećenje funkcija

```
#include <iostream>
using namespace std;
// Preopterećena funkcija za ispis cijelih brojeva
void Ispis(int broj) {
    cout << "Cijeli broj: " << broj << endl;
}
// Preopterećena funkcija za ispis decimalnih brojeva
void Ispis(double broj) {
    cout << "Decimalni broj: " << broj << endl;
}
int main() {
    Ispis(5);          // Poziv funkcije za cijele brojeve
    Ispis(3.14);       // Poziv funkcije za decimalne brojeve
    return 0;
}
```

Funkcija s pokazivačem na polje

```
#include <iostream>
using namespace std;
// Funkcija za pronađak maksimalne vrijednosti u nizu
int MaksimalnaVrijednost(int* niz, int velicina) {
    int max = niz[0];
    for (int i = 1; i < velicina; i++) {
        if (niz[i] > max) {
            max = niz[i];
        }
    }
    return max;
}
int main() {
    int niz[] = {3, 5, 7, 2, 8};
    int velicina = sizeof(niz) / sizeof(niz[0]);
    // Poziv funkcije
    cout << "Najveći element u nizu" << MaksimalnaVrijednost(niz, velicina);
    return 0;
}
```

Funkcija s referencama

```
#include <iostream>
using namespace std;
// Funkcija za zamjenu vrijednosti dviju varijabli
void Zamjeni(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}
int main() {
    int x = 10, y = 20;
    cout << "Prije zamjene: x = " << x << ", y = " << y << endl;
    Zamjeni(x, y);
    cout << "Nakon zamjene: x = " << x << ", y = " << y << endl;
    return 0;
}
```

11. Funkcije (2. dio)

Parametri funkcije mogu imati početne vrijednosti, što omogućuje pozivanje funkcije s manje argumenata. Početni parametri nazivaju se i repni parametri.

- Svi parametri nakon prvog parametra s početnom vrijednošću također moraju imati početne vrijednosti.
- Argumenti i parametri se uparuju **redoslijedom**.

```
void HorizontLinija(int brZvjezdica = 10, bool prazanRed = true) {  
    for (int i = 1; i <= brZvjezdica; i++) cout << "*";  
    if (prazanRed) cout << "\n";  
}  
// Pozivi funkcije:  
HorizontLinija();           // Ispisuje 10 zvjezdica  
HorizontLinija(5, false);   // Ispisuje 5 zvjezdica bez praznog reda
```

Prenošenje argumenata funkcijama

- Kopija argumenta prosleđuje se funkciji.
- Promjene unutar funkcije ne utječu na originalnu varijablu.

```
void Prosljedi(int var) {  
    var = 10; // Promjena vrijedi samo unutar funkcije  
}  
int broj = 5;  
Prosljedi(broj);  
cout << broj; // Ispisuje: 5
```

Po referenci: Funkcija prima referencu na varijablu, Promjene utječu na originalnu varijablu.

```
void Prosljedi(int& var) {  
    var = 10; // Promjena utječe na originalnu varijablu  
}  
int broj = 5;  
Prosljedi(broj);  
cout << broj; // Ispisuje: 10
```

Pokazivači:

Funkcija prima adresu varijable.

```
void Prosljedi(int* var) {  
    *var = 10; // Dereferenciranjem mijenjamo originalnu varijablu  
}  
int broj = 5;  
Prosljedi(&broj);  
cout << broj; // Ispisuje: 10
```

Koncept	Prosljeđivanje po vrijednosti	Prosljeđivanje po referenci
Kako se prenosi?	Kopira se vrijednost argumenta.	Prenosi se referenca na varijablu.
Utječe na original?	Ne utječe na original.	Da – originalna varijabla se mijenja.
Primjena?	Kad ne želimo mijenjati podatke.	Kad trebamo mijenjati originalnu vrijednost.

Rekurzivne funkcije

Rekurzija je tehnika u kojoj funkcija poziva samu sebe kako bi rješila problem. Ključni dijelovi rekurzivne funkcije su:

1. **Baza rekurzije (bazni slučaj)** – uvjet kada se rekurzija zaustavlja.
2. **Rekurzivni slučaj** – funkcija poziva samu sebe s manjim podproblemom

Primjer rekurzivne funkcije: Faktorijel broja

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

int faktorijel(int n) {
    if (n == 0) return 1; // Bazni slučaj
    return n * faktorijel(n - 1); // Rekurzivni slučaj
}

int main() {
    int broj = 5;
    cout << "Faktorijel od " << broj << " je: " << faktorijel(broj)
    << endl;
    return 0;
}
```

Objasnimo **kako radi**:

- Ako $n == 0$, vraća se 1, što je bazni slučaj.
- Inače, funkcija se poziva s $n-1$ sve dok se ne dostigne bazni slučaj.

Rekurzija i Prosljeđivanje Polja

Rekurzivna funkcija može koristiti prosljeđivanje polja za probleme poput zbrajanja elemenata niza.

Primjer zbrajanja elemenata niza rekurzivno:

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

int sumaPolja(int polje[], int velicina) {
    if (velicina == 0) return 0; // Bazni slučaj
    return polje[velicina - 1] + sumaPolja(polje, velicina - 1); // Rekurzivni
    slučaj
}

int main() {
    int brojevi[] = {1, 2, 3, 4, 5};
    cout << "Suma polja je: " << sumaPolja(brojevi, 5) << endl;
    return 0;
}
```

Kako funkcioniра:

- Bazni slučaj: ako je veličina polja 0, vrati 0.
- Inače, rekurzivno poziva samu sebe smanjujući veličinu polja za 1 i zbraja elemente.

11.1. Osnovni primjeri

Suma prvih nnn prirodnih brojeva

```
int SumaN(int n) {
    if (n <= 0) return 0; // Osnovni slučaj
    return n + SumaN(n - 1); // Rekursivni poziv
}

cout << SumaN(5); // Ispisuje: 15 (1+2+3+4+5)
```

Primjer za izračunavanje NZD (najvećeg zajedničkog djelitelja):

```
//općenito
int NZD(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

//rekurzivno
int NZD(int a, int b) {
    if (b == 0) return a;
    return NZD(b, a % b);
}
```

CString funkcija KORISNO

```
//DUŽINA NIZA
char niz[] = "Programiranje";
cout << strlen(niz); // Ispisuje: 13

//KOPIRANJE NIZA
char izvor[] = "C++";
char odredisni[10];
strcpy(odredisni, izvor);
cout << odredisni; // Ispisuje: C++

//SPAJANJE NIZOVA
char niz1[20] = "Pozdrav";
char niz2[] = " svijete!";
strcat(niz1, niz2);
cout << niz1; // Ispisuje: Pozdrav svijete!
```

11.2. Primjeri naprednih zadataka

Tri najveća elementa u nizu //nije rekurzija

```
void MaksTri(int niz[], int vel, int& prvi, int& drugi, int& treci) {  
    prvi = drugi = treci = INT_MIN;  
    for (int i = 0; i < vel; i++) {  
        if (niz[i] > prvi) {  
            treci = drugi;  
            drugi = prvi;  
            prvi = niz[i];  
        } else if (niz[i] > drugi) {  
            treci = drugi;  
            drugi = niz[i];  
        } else if (niz[i] > treci) {  
            treci = niz[i];  
        }  
    }  
}
```

Rekurzivni algoritam za testiranje relativne prostosti

Opis: Dva broja su relativno prosta ako je njihov najveći zajednički djelitelj (NZD) jednak 1. Napravite rekurzivni algoritam koji će testirati i ispisivati informaciju o tome jesu li brojevi relativno prosti.

```
#include <iostream>  
using namespace std;  
  
// Funkcija za rekurzivni izračun NZD (Euklidov algoritam)  
int nzd(int a, int b) {  
    if (b == 0)  
        return a; // Bazni slučaj  
    return nzd(b, a % b); // Rekurzivni poziv s manjim brojevima  
}  
// Funkcija za provjeru relativne prostosti  
bool suRelativnoProsti(int a, int b) {  
    return nzd(a, b) == 1;  
}  
int main() {  
    int broj1, broj2;  
    cout << "Unesite dva broja: ";  
    cin >> broj1 >> broj2;  
  
    if (suRelativnoProsti(broj1, broj2))  
        cout << broj1 << " i " << broj2 << " su relativno prosti.\n";  
    else  
        cout << broj1 << " i " << broj2 << " nisu relativno prosti.\n";  
  
    return 0;  
}
```

Objašnjenje:

- Koristi se Euklidov algoritam za izračun najvećeg zajedničkog djelitelja (NZD).
- Ako je NZD dva broja jednak 1, brojevi su relativno prosti.

Rekurzija za traženje parova brojeva i njihovih potencija koji daju zadani broj

Opis: Pronaći parove brojeva (a, b) gdje je $a^b = \text{zadani broj}$.

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

// Rekursivna funkcija za traženje parova potencija
void pronadiParove(int broj, int baza = 2) {
    if (baza > broj) return; // Bazni slučaj, zaustavljamo rekurziju

    // Provjera potencije baze koja daje broj
    int eksponent = 0;
    int rezultat = 1;
    while (rezultat < broj) {
        rezultat *= baza;
        eksponent++;
    }

    // Ako je rezultat jednak broju, ispisujemo par
    if (rezultat == broj)
        cout << baza << "^" << eksponent << " = " << broj << endl;

    // Rekursivni poziv s povećanom bazom
    pronadiParove(broj, baza + 1);
}

int main() {
    int broj;
    cout << "Unesite broj: ";
    cin >> broj;

    cout << "Parovi potencija za broj " << broj << " su: \n";
    pronadiParove(broj);

    return 0;
}
```

Objašnjenje:

- Rekursivno pretražuje baze od 2 pa nadalje.
- Za svaku bazu izračunava potencije dok ne dostigne zadani broj.
- Ako je potencija jednaka zadanom broju, ispisuje se par.

Rekurzija s palindromom bez korištenja string biblioteke

Opis: Provjera je li broj palindrom (čita se isto sprijeda i straga).

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

// Funkcija za dobivanje broja znamenki
int brojZnamenki(int n) {
    if (n < 10) return 1;
    return 1 + brojZnamenki(n / 10);
}

// Rekursivna funkcija za provjeru palindroma
bool jePalindrom(int n, int duzina) {
    if (n < 10) return true; // Bazni slučaj

    int prvaZnamenka = n / pow(10, duzina - 1);
    int zadnjaZnamenka = n % 10;

    if (prvaZnamenka != zadnjaZnamenka) return false;

    // Uklanjanje prve i zadnje znamenke
    int noviBroj = (n % static_cast<int>(pow(10, duzina - 1))) / 10;

    // Rekursivni poziv sa smanjenim brojem znamenki
    return jePalindrom(noviBroj, duzina - 2);
}

int main() {
    int broj;
    cout << "Unesite broj: ";
    cin >> broj;

    if (jePalindrom(broj, brojZnamenki(broj)))
        cout << broj << " je palindrom.\n";
    else
        cout << broj << " nije palindrom.\n";

    return 0;
}
```

Objašnjenje:

- Broj se rekursivno uspoređuje tako što se prva i zadnja znamenka odsjecaju.
- Ako su sve znamenke jednake u ogledalu, broj je palindrom.

Dvostruko vezana lista za parne i neparne brojeve

Opis: Implementacija dvostruko vezane liste gdje se parni brojevi pohranjuju u jednu, a neparni u drugu listu.

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

// Struktura za čvor liste
struct Cvor {
    int podatak;
    Cvor* sljedeci;
    Cvor(int vrijednost) : podatak(vrijednost), sljedeci(nullptr) {}
};

// Funkcija za dodavanje elementa na kraj liste
void dodajUCvor(Cvor*& glava, int vrijednost) {
    Cvor* novi = new Cvor(vrijednost);
    if (!glava) {
        glava = novi;
    } else {
        Cvor* temp = glava;
        while (temp->sljedeci) {
            temp = temp->sljedeci;
        }
        temp->sljedeci = novi;
    }
}

// Funkcija za ispis liste
void ispisiListu(Cvor* glava) {
    while (glava) {
        cout << glava->podatak << " -> ";
        glava = glava->sljedeci;
    }
    cout << "null\n";
}

// Rekursivna funkcija za razdvajanje brojeva u parne i neparne liste
void razdvojiParneINeparne(Cvor* glava, Cvor*& parni, Cvor*& neparni) {
    if (!glava) return; // Bazni slučaj: ako nema više elemenata, zaustavi
    // rekurziju

    // Provjera je li broj paran ili neparan
    if (glava->podatak % 2 == 0)
        dodajUCvor(parni, glava->podatak); // Dodaj u listu parnih brojeva
    else
        dodajUCvor(neparni, glava->podatak); // Dodaj u listu neparnih brojeva

    // Rekursivni poziv za sljedeći element liste
    razdvojiParneINeparne(glava->sljedeci, parni, neparni);
}

//nastavak na sljedećoj stranici
```

```
// Glavna funkcija
int main() {
    Cvor* lista = nullptr;
    Cvor* parni = nullptr;
    Cvor* neparni = nullptr;

    // Dodavanje brojeva u početnu listu
    dodajUCvor(lista, 1);
    dodajUCvor(lista, 2);
    dodajUCvor(lista, 3);
    dodajUCvor(lista, 4);
    dodajUCvor(lista, 5);

    cout << "Originalna lista: ";
    ispisiListu(lista);

    // Razdvajanje brojeva u parne i neparne liste
    razdvojiParneINeparne(lista, parni, neparni);

    // Ispis rezultata
    cout << "\nParni brojevi: ";
    ispisiListu(parni);
    cout << "Neparni brojevi: ";
    ispisiListu(neparni);

    return 0;
}
```

12. Podjeli pa ovladaj

Dizajn algoritma PODJELI PA OVLADAJ temeljen na **razdvajanju problema u manje podprobleme**, **rješavanju tih podproblema** i **spajanju rješenja** u konačno rješenje. Često koristi rekurziju za razlaganje problema.

Koraci metode "Podjeli pa ovladaj"

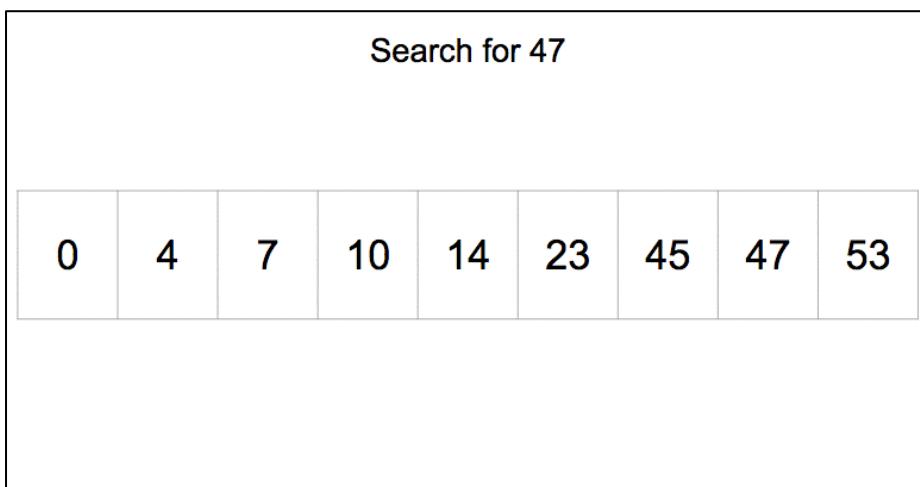
1. **Podjeli:** Problem se razlaže na manje, nezavisne podprobleme.
2. **Ovladaj:** Svaki podproblem se rješava rekurzivno ili direktno ako je trivijalan.
3. **Spoji:** Rješenja podproblema kombiniraju se u rješenje izvornog problema.

Primjeri algoritama:

Algoritam	Prednosti	Nedostaci
Quicksort	Brz, jednostavna implementacija	Loša efikasnost u najgorem slučaju ($O(n^2)$)
Merge Sort	Stabilan, efikasan za velike nizove	Zahtijeva dodatnu memoriju
Binarno pretraživanje	Izuzetno brzo za sortirane nizove	Niz mora biti sortiran

1. **Binarno pretraživanje:** Efikasnost: $O(\log n)$.

- Pretražuje sortiran niz dijeljenjem na polovice



<https://materijali.xfer.hr/docs/sortiranje-i-pretrazivanje/binarno-pretrazivanje/>

```
int BinarnoPretrazivanje(int niz[], int lijevo, int desno, int kljuc) {  
    while (lijevo <= desno) {  
        int sredina = lijevo + (desno - lijevo) / 2;  
        if (niz[sredina] == kljuc) return sredina;  
        else if (kljuc > niz[sredina]) lijevo = sredina + 1;  
        else desno = sredina - 1;  
    }  
    return -1; // Element nije pronađen  
}
```

2. **Quicksort:** Prosječna efikasnost: $O(n \log n)$.

- Sortira niz odabirom pivota, dijeljenjem elemenata na manje i veće od pivota te rekursivnim sortiranjem.

Koraci algoritma:

- Odaber pivot (stožerni element).
- Podijeli elemente u dvije grupe:
 - Manji od pivota.
 - Veći od pivota.
- Rekursivno sortiraj obje grupe.



6 5 3 1 8 7 2 4

<https://commons.wikimedia.org/wiki/File:Quicksort-example.gif>

```
void Quicksort(int niz[], int donja, int gornja) {  
    if (gornja <= donja) return;  
  
    int pivot = niz[donja]; // Stožerni element  
    int i = donja, j = gornja + 1;  
  
    while (true) {  
  
        while (niz[++i] < pivot) if (i == gornja) break;  
        while (pivot < niz[--j]) if (j == donja) break;  
        if (i >= j) break;  
        swap(niz[i], niz[j]);  
    }  
  
    swap(niz[donja], niz[j]);  
    Quicksort(niz, donja, j - 1);  
    Quicksort(niz, j + 1, gornja);  
}
```

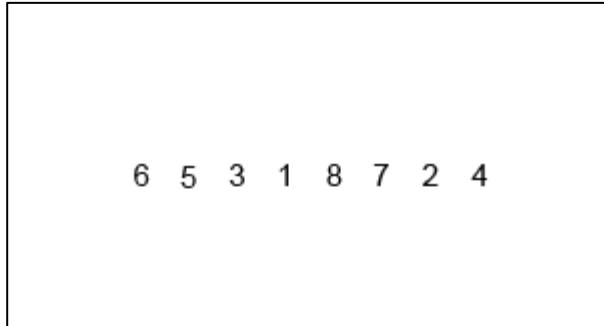
3. Merge Sort (Sortiranje spajanjem):

Najgora efikasnost: $O(n \log n)$

- Dijeli niz na dva dijela, sortira ih i spaja u konačni rezultat

Koraci algoritma:

- Podijeli niz na dva podniza.
- Rekurzivno sortiraj svaki podniz.
- Spoji dva sortirana podniza u jedan.



6 5 3 1 8 7 2 4

<https://en.m.wikipedia.org/wiki/File:Merge-sort-example-300px.gif>

```
void Spajanje(int niz[], int donja, int sredina, int gornja, int pomocno[]) {  
    int i = donja, j = sredina + 1;  
  
    for (int k = donja; k <= gornja; k++) pomocno[k] = niz[k];  
  
    for (int k = donja; k <= gornja; k++) {  
  
        if (i > sredina) niz[k] = pomocno[j++];  
  
        else if (j > gornja) niz[k] = pomocno[i++];  
  
        else if (pomocno[j] < pomocno[i]) niz[k] = pomocno[j++];  
  
        else niz[k] = pomocno[i++];  
  
    }  
}  
  
void MergeSort(int niz[], int donja, int gornja, int pomocno[]) {  
    if (gornja <= donja) return;  
  
    int sredina = donja + (gornja - donja) / 2;  
  
    MergeSort(niz, donja, sredina, pomocno);  
  
    MergeSort(niz, sredina + 1, gornja, pomocno);  
  
    Spajanje(niz, donja, sredina, gornja, pomocno);  
}
```

PRIMJERI:

Kreirajte cijelobrojno polje od 12 elemenata. U polje generirajte nasumične cijele brojeve u rastućem redoslijedu na način da svaki idući broj bude veći od prethodnog za nasumičnu vrijednost u rasponu od 1 do 5. Ispišite cijelo polje u jednom retku, brojevi odvojeni razmacima. Zatražite od korisnika unos tražene vrijednosti. Kreirajte potprogram u kojem ćete implementirati **rekurzivno binarno pretraživanje**.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;
// Rekurzivna funkcija za binarno pretraživanje
int BinarnoPretrazivanje(int niz[], int lijevo, int desno, int trazeni) {
    if (lijevo > desno) return -1; // Ako nije pronađen
    int sredina = lijevo + (desno - lijevo) / 2;
    if (niz[sredina] == trazeni) return sredina; // Pronađena vrijednost
    else if (niz[sredina] > trazeni) return BinarnoPretrazivanje(niz, lijevo, sredina - 1, trazeni); // Traži lijevo
    else return BinarnoPretrazivanje(niz, sredina + 1, desno, trazeni); // Traži desno
}
int main() {
    srand(time(0)); // Inicijalizacija generatora slučajnih brojeva
    const int VELICINA = 12;
    int niz[VELICINA];
    // Generiranje polja s rastućim vrijednostima
    niz[0] = rand() % 10 + 1; // Prvi broj od 1 do 10
    for (int i = 1; i < VELICINA; i++) {
        niz[i] = niz[i - 1] + (rand() % 5 + 1); // Svaki broj veći od prethodnog za 1
    }
    cout << "Polje: ";
    for (int i = 0; i < VELICINA; i++) {
        cout << niz[i] << " ";
    }
    cout << endl;
    int trazeni;
    cout << "Unesite broj za pretragu: ";
    cin >> trazeni;
    // Poziv funkcije za binarno pretraživanje
    int indeks = BinarnoPretrazivanje(niz, 0, VELICINA - 1, trazeni);
    // Rezultat pretrage
    if (indeks != -1)
        cout << "Broj " << trazeni << " pronađen na indeksu " << indeks << "." << endl;
    else
        cout << "Broj " << trazeni << " nije pronađen u polju." << endl;
    return 0;
}
```

Kreirajte cijelobrojno polje od 15 elemenata. U polje generirajte nasumične vrijednosti u rasponu od 50 do 150. Ispišite cijelo polje u jednom retku, brojevi odvojeni razmacima. Kreirajte potprogram koji polje sortira **uzlazno** metodom brzog sortiranja (**quicksort**).

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int Particija(int niz[], int donja, int gornja) { // za partitioniranje niza
    int pivot = niz[gornja]; // Pivot je posljednji element
    int i = donja - 1; // Indeks manjeg elementa
    for (int j = donja; j < gornja; j++) {
        if (niz[j] <= pivot) { // Ako je trenutni element manji ili jednak pivotu
            i++;
            swap(niz[i], niz[j]); // Zamjena elemenata
        }
    }
    swap(niz[i + 1], niz[gornja]); // Premještanje pivota na točno mjesto
    return i + 1; // Vraća indeks pivota
}

void QuickSort(int niz[], int donja, int gornja) { //Rekurzivna za quicksort
    if (donja < gornja) {
        int pivotIndeks = Particija(niz, donja, gornja); // Podjela niza
        QuickSort(niz, donja, pivotIndeks - 1); // Sortiranje lijevog dijela
        QuickSort(niz, pivotIndeks + 1, gornja); // Sortiranje desnog dijela
    }
}

//--nastavak na sljedećoj stranici--
```

```
int main() {
    srand(time(0)); // Postavlja generator nasumičnih brojeva
    const int VELICINA = 15;
    int niz[VELICINA];
    // Generiranje nasumičnih vrijednosti u rasponu [50, 150]
    for (int i = 0; i < VELICINA; i++) {
        niz[i] = rand() % 101 + 50; // Brojevi od 50 do 150
    }
    // Ispis nesortiranog niza
    cout << "Nesortirano polje: ";
    for (int i = 0; i < VELICINA; i++) {
        cout << niz[i] << " ";
    }
    cout << endl;
    // Sortiranje polja
    QuickSort(niz, 0, VELICINA - 1);
    // Ispis sortiranog niza
    cout << "Sortirano polje (uzlazno): ";
    for (int i = 0; i < VELICINA; i++) {
        cout << niz[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Kreirajte cijelobrojno polje od 12 elemenata. Omogućite korisniku unos vrijednosti u polje. Kreirajte potprogram koji polje sortira uzlazno metodom spajanja (**mergesort**).

```
#include <iostream>

using namespace std;

// Funkcija za spajanje dvaju podnizova

void Spajanje(int niz[], int lijevo, int sredina, int desno) {

    int i = lijevo, j = sredina + 1, k = 0;

    int pomocno[desno - lijevo + 1];

    // Spajanje dva sortirana dijela niza

    while (i <= sredina && j <= desno) {

        if (niz[i] <= niz[j]) {

            pomocno[k++] = niz[i++];

        } else {

            pomocno[k++] = niz[j++];

        }

    }

    // Dodavanje preostalih elemenata iz lijevog dijela

    while (i <= sredina) {

        pomocno[k++] = niz[i++];

    }

    // Dodavanje preostalih elemenata iz desnog dijela

    while (j <= desno) {

        pomocno[k++] = niz[j++];

    }

    // Kopiranje spojenog niza natrag u originalni niz

    for (int l = 0; l < k; l++) {

        niz[lijevo + l] = pomocno[l];

    }

}
```

```

// Rekurzivna funkcija za sortiranje

void MergeSort(int niz[], int lijevo, int desno) {
    if (lijevo < desno) {
        int sredina = lijevo + (desno - lijevo) / 2;
        // Rekurzivno sortiranje lijevog i desnog dijela
        MergeSort(niz, lijevo, sredina);
        MergeSort(niz, sredina + 1, desno);
        // Spajanje sortiranja
        Spajanje(niz, lijevo, sredina, desno);
    }
}

int main() {
    const int VELICINA = 12;
    int niz[VELICINA];
    cout << "Unesite 12 cijelih brojeva: " << endl; // Unos elemenata polja
    for (int i = 0; i < VELICINA; i++) {
        cin >> niz[i];
    }
    cout << "Nesortirano polje: " // Ispis nesortiranog polja
    for (int i = 0; i < VELICINA; i++) {
        cout << niz[i] << " ";
    }
    cout << endl; // Sortiranje polja
    MergeSort(niz, 0, VELICINA - 1);
    cout << "Sortirano polje (uzlazno): " // Ispis sortiranog polja
    for (int i = 0; i < VELICINA; i++) {
        cout << niz[i] << " ";
    }
    cout << endl;
    return 0;
}

```

13. Tekstualne datoteke

Uvod u tekstualne datoteke

- **Motivacija:**
 - Podaci koji se obrađuju u glavnoj memoriji nestaju nakon završetka programa.
 - Tekstualne datoteke omogućuju trajno spremanje podataka za buduću obradu.
- **Upotreba:**
 - Čitanje i pisanje podataka iz/na datoteke za spremanje, analizu ili prijenos podataka između aplikacija.

Osnovni pojmovi

1. Biblioteke za rad s datotekama:

- **<fstream>**: Sadrži naredbe za rad s datotekama.
- **<iostream>**: Koristi se za ulaz i izlaz podataka (poput cin i cout).

2. Vrste tokova podataka:

- **Ulazni tok (ifstream)**: Čita podatke iz datoteke.
- **Izlazni tok (ofstream)**: Piše podatke u datoteku.
- **Ulazno-izlazni tok (fstream)**: Omogućuje istovremeno čitanje i pisanje podataka.

Rukovanje datotekama

Ulazne datoteke:

```
ifstream ulaz("ime_datoteke.txt");
```

Izlazne datoteke:

```
ofstream izlaz("ime_datoteke.txt");
```

Čitanje podataka

Čitanje redaka

```
char linija[20000];
while (ulaz.getline(linija, 20000)) {
    cout << linija << endl;
}
```

Zatvaranje datoteke

Nakon završetka rada

```
ulaz.close();
```

Zadatak: Prebrojavanje slova u datoteci**• Problem:**

- Prebrojati koliko se puta svako slovo pojavljuje u datoteci (mala i velika slova se tretiraju jednako).
- Sortirati slova prema broju pojavljivanja u silaznom redoslijedu.

```
#include <iostream>
```

```
#include <fstream>
```

```
#include <cstring>
```

```
using namespace std;
```

```
// Slog za učestalost slova
```

```
struct Slovo {
```

```
    char znak;
```

```
    int broj;
```

```
};
```

```
// Funkcija za inicijalizaciju polja slova
```

```
void Inicijaliziraj(Slovo slova[]) {
```

```
    for (int i = 0; i < 26; i++) {
```

```
        slova[i].znak = i + 'A';
```

```
        slova[i].broj = 0;
```

```
    }
```

```
}
```

```
// Funkcija za sortiranje umetanja
```

```
void Sortiraj(Slovo slova[]) {
```

```
    for (int i = 1; i < 26; i++) {
```

```
        Slovo temp = slova[i];
```

```
        int j = i - 1;
```

```
        while (j >= 0 && slova[j].broj < temp.broj) {
```

```
            slova[j + 1] = slova[j];
```

```
            j--;
```

```
        }
```

```
        slova[j + 1] = temp;
```

```
    }
```

```
}
```

```

// Funkcija za obradu datoteke
void ObradiDatoteku(const char* imeDatoteke, Slovo slova[]) {
    ifstream ulaz(imeDatoteke);
    if (!ulaz) {
        cout << "Datoteka ne postoji!" << endl;
        exit(1);
    }
    char linija[20000];
    while (ulaz.getline(linija, 20000)) {
        int duljina = strlen(linija);
        for (int i = 0; i < duljina; i++) {
            if (linija[i] >= 'A' && linija[i] <= 'Z') {
                slova[linija[i] - 'A'].broj++;
            } else if (linija[i] >= 'a' && linija[i] <= 'z') {
                slova[linija[i] - 'a'].broj++;
            }
        }
    }
    ulaz.close();
}
// Funkcija za ispis rezultata
void Ispisi(Slovo slova[]) {
    for (int i = 0; i < 26; i++) {
        if (slova[i].broj > 0) {
            cout << slova[i].znak << ":" << slova[i].broj << endl;
        }
    }
}
// Glavni program
int main() {
    char imeDatoteke[50];
    Slovo slova[26];
    // Inicijalizacija
    Inicijaliziraj(slova);
    // Unos imena datoteke
    cout << "Unesite ime datoteke: ";
    cin.getline(imeDatoteke, 50);
    // Obrada datoteke
    ObradiDatoteku(imeDatoteke, slova);
    // Sortiranje rezultata
    Sortiraj(slova);
    // Ispis rezultata
    Ispisi(slova);

    return 0;
}

```

14. Binarne datoteke

Definicija:

- Binarne datoteke pohranjuju podatke u binarnom formatu, a ne u čitljivom tekstualnom obliku.
- Sadrže slogove podataka s fiksnom duljinom, što omogućuje efikasno pretraživanje i pristup podacima.

Razlika od tekstualnih datoteka:

- Tekstualne datoteke pohranjuju podatke kao niz znakova (ASCII format).
- Binarne datoteke direktno pohranjuju podatke iz memorije (binarnim formatom), što omogućuje brži pristup.

Zašto koristiti binarne datoteke?

- **Prednosti:** Manja veličina datoteke. Brže čitanje i pisanje. Direktan pristup slogovima pomoću kurzora (seekg, seekp).
- **Nedostaci:** Podaci nisu čitljivi za korisnika bez specifičnih alata. Ovisni su o strukturi slogova.

Rad s binarnim datotekama u C++

Struktura za slogove:

- Podaci se organiziraju u slogove (npr. studenti, predmeti).

```
struct Student {  
    unsigned int indeks;  
    char ime[30], prezime[30];  
    char godina;  
};
```

Radnje:

Otvaranje datoteka:

- Koristi se fstream, ifstream ili ofstream s opcijama:
 - ios::binary – za rad u binarnom modu.
 - ios::app – za dodavanje na kraj datoteke.
 - ios::in, ios::out – za čitanje ili pisanje.

Primjer otvaranja datoteke za pisanje:

```
ofstream datoteka("studenti.dat", ios::binary | ios::app);
```

Pisanje u binarnu datoteku: Koristi se metoda write.

```
Student s;  
  
ofstream stud("studenti.dat", ios::binary | ios::app);  
  
stud.write((char *)&s, sizeof(s));  
  
stud.close();
```

Citanje iz binarne datoteke: Koristi se metoda read.

```
Student s;  
  
ifstream stud("studenti.dat", ios::binary);  
  
while (stud.read((char *)&s, sizeof(s))) {  
  
    cout << "Ime: " << s.ime << ", Prezime: " << s.prezime << endl;  
}  
  
stud.close();
```

Kursori u datotekama:

- seekg – pomicanje kursora za čitanje.
- seekp – pomicanje kursora za pisanje.
- tellg – dohvati trenutne pozicije kursora za čitanje.
- tellp – dohvati trenutne pozicije kursora za pisanje.

Primjer

Primjer programa za **unos i pretraživanje** studenata

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
// Struktura za studenta
struct Student {
    unsigned int indeks;
    char ime[30], prezime[30];
    char godina;
};
void Unos() { // Funkcija za unos studenata
    Student s;
    char dn;
    ofstream stud("studenti.dat", ios::binary | ios::app);
    do {
        cout << "Broj indeksa: "; cin >> s.indeks;
        cout << "Prezime: "; cin >> s.prezime;
        cout << "Ime: "; cin >> s.ime;
        cout << "Godina studija: "; cin >> s.godina;
        // Pisanje u datoteku
        stud.write((char *)&s, sizeof(s));
        cout << "Zelite li jos unosa? (d/n): ";
        cin >> dn;
    } while (dn == 'd' || dn == 'D');
    stud.close();
}
```

Kod za pretraživanje studenta po broju indeksa:

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

// Funkcija za pretraživanje studenta
void Pretrazi(unsigned int indeks) {
    Student s;
    ifstream stud("studenti.dat", ios::binary);
    bool nadjeno = false;
    while (stud.read((char *)&s, sizeof(s))) {
        if (s.indeks == indeks) {
            nadjeno = true;
            cout << "Student pronađen!" << endl;
            cout << "Ime: " << s.ime << ", Prezime: " << s.prezime << ", Godina: "
            << s.godina << endl;
            break;
        }
    }
    if (!nadjeno) {
        cout << "Student s indeksom " << indeks << " nije pronađen." << endl;
    }
    stud.close();
}
```

Glavna funkcija s izbornikom:

```
int main() {  
  
    char izbor;  
  
    do {  
  
        cout << "1. Unos studenata" << endl;  
        cout << "2. Pretrazivanje studenata" << endl;  
        cout << "0. Izlaz" << endl;  
        cout << "Izbor: ";  
        cin >> izbor;  
  
        switch (izbor) {  
            case '1':  
                Unos();  
                break;  
            case '2': {  
                unsigned int indeks;  
                cout << "Unesite broj indeksa: ";  
                cin >> indeks;  
                Pretrazi(indeks);  
                break;  
            }  
        }  
    } while (izbor != '0');  
  
    return 0;  
}
```

Kreirajte binarnu datoteku u kojoj će se pohraniti 30 cijelih brojeva generiranih slučajno u rasponu od 100 do 999. Zapišite te brojeve u binarnu datoteku i ispišite njihov sadržaj (brojevi u jednom retku, odvojeni razmakom). Nakon toga, implementirajte funkciju za pronalaženje najmanjeg i najvećeg broja u datoteci, te ispišite njihove vrijednosti i pozicije u datoteci.

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace std;

// Funkcija za generiranje i zapisivanje brojeva u binarnu datoteku
void GenerirajDatoteku(const char* imeDatoteke) {
    ofstream datoteka(imeDatoteke, ios::binary);
    if (!datoteka) {
        cout << "Greška pri otvaranju datoteke!" << endl;
        exit(1);
    }
    srand(time(0)); // Inicijalizacija generatora slučajnih brojeva
    for (int i = 0; i < 30; i++) {
        int broj = rand() % 900 + 100; // Generira broj u rasponu [100, 999]
        datoteka.write((char*)&broj, sizeof(broj));
    }
    datoteka.close();
}

void IspisiDatoteku(const char* imeDatoteke) { // za ispis sadržaja binarne datoteke
    ifstream datoteka(imeDatoteke, ios::binary);
    if (!datoteka) { cout << "Greška pri otvaranju datoteke!" << endl; exit(1); }
    int broj;
    while (datoteka.read((char*)&broj, sizeof(broj))) {
        cout << broj << " "; } cout << endl;
    datoteka.close(); }
```

```

// Funkcija za pronađak najmanjeg i najvećeg broja

void PronadiMinIMax(const char* imeDatoteke) {

    ifstream datoteka(imeDatoteke, ios::binary);

    if (!datoteka) {

        cout << "Greška pri otvaranju datoteke!" << endl;
        exit(1);
    }

    int broj, minBroj = INT_MAX, maxBroj = INT_MIN;
    int minPozicija = -1, maxPozicija = -1;
    int pozicija = 0;

    while (datoteka.read((char*)&broj, sizeof(broj))) {

        if (broj < minBroj) {

            minBroj = broj;
            minPozicija = pozicija;
        }

        if (broj > maxBroj) {

            maxBroj = broj;
            maxPozicija = pozicija;
        }

        pozicija++;
    }

    datoteka.close();

    // Ispis rezultata

    cout << "Najmanji broj: " << minBroj << " (pozicija: " << minPozicija << ")" << endl;

    cout << "Najveći broj: " << maxBroj << " (pozicija: " << maxPozicija << ")" << endl;
}

```

```
int main() {  
    const char* imeDatoteke = "brojevi.dat";  
    // Generiranje i zapisivanje brojeva u binarnu datoteku  
    GenerirajDatoteku(imeDatoteke);  
    // Ispis sadržaja datoteke  
    IspisiDatoteku(imeDatoteke);  
    // Pronalaženje i ispis najmanjeg i najvećeg broja  
    PronadiMinIMax(imeDatoteke);  
    return 0;  
}
```

15. Popis naredba

Popis komanda s opis i bibliotekama (kopirano od ProgDemos)

DATOTEKE		
komanda	OPIS	BIBLIOTEKA
ofstream	poput cout-a koji šalje podatke u konzolu (na zaslon), ofstream šalje podatke u datoteku	fstream
ifstream	poput cin-a "s tipkovnice", učitava podatke iz datoteke; vraća 0 ako datoteka ne postoji	fstream
fstream	osnovni datotečni tok podataka (ofstream i ifstream su njegovi određeni oblici)	fstream
datoteka.open(naziv, mod1 mod2 mod3)	otvori datoteku "naziv", ako korisitmo mod ios::binary, otvaramo binarnu datoteku	fstream
datoteka.getline()	radi kao tipičan getline, samo s podacima (retcima) iz tekstualne datoteke	fstream
datoteka.eof()	detektira kraj tekstualne datoteke, nije pouzdano (bolje koristiti !datoteka.getline() u while petlji)	fstream
datoteka.write((char *)&upis, sizeof(upis))	upisuje u binarnu datoteku saržaj varijable "upis"	fstream
datoteka.read((char *)&upis, sizeof(upis))	čita iz binarne datoteke sadržaj u varijablu "upis"	fstream
seekg	postavi vrijednost kursora za čitanje (get pointer) u binarnu datoteku	fstream
seekp	postavi vrijednost kursora za upis (put pointer) u binarnu datoteku	fstream
tellg	vraća "lokaciju" (u bajtovima) kursora za čitanje u binarnoj datoteci	fstream
tellp	vraća "lokaciju" (u bajtovima) kursora za upis u binarnoj datoteci	fstream

...nastavak na drugoj stranici....

OSTALO

komanda	OPIS	BIBLIOTEKA
<code>cin >></code>	upis podataka (čitanje s tipkovnice)	<code>iostream</code>
<code>cout <<</code>	ispis podataka (u konzolu/na zaslon)	<code>iostream</code>
<code>setprecision(n)</code>	ispisat će se n znamenaka decimalnog broja	<code>iomanip</code>
<code>if(uvjet) {naredbe}</code> <code>else{naredbe}</code>	osnovna selekcija	
<code>switch (varijabla) {case</code> <code>vrijednost: naredbe}</code>	provjerava vrijednost varijable	
<code>break</code>	izlaz iz petlje (ili switcha)	
<code>continue</code>	prebac se na početak (glavu) petlje	
<code>pow(x,y)</code>	x^y	<code>cmath</code>
<code>sqrt(x)</code>	\sqrt{x}	<code>cmath</code>
<code>logički izraz ? istina</code> <code>: laž</code>	ternarni operator	
<code>for (inicijalizacija;</code> <code>uvjet; promjena) {}</code>	for iteracija	
<code>while (uvjet) {}</code>	while iteracija	
<code>do{naredbe}while(uvjet)</code>	do-while iteracija	
<code>time(0)</code>	vraća trenutačno vrijeme (broj sekundi od 1970.)	<code>ctime</code>
<code>srand() //često</code> <code>koristeno:</code> <code>srand(time(0));</code>	postavi random seed za slučajnije random (pseudoslučajne) brojeve	<code>cstdlib</code>
<code>rand()</code>	vraća random broj od 0 do 32767 (granica nije nužno tako fiksna, ali ovo je C++ standard)	<code>cstdlib</code>
<code>cin.get()</code>	poput cin-a, ali pamti i razmak; rađe uvijek koristi cin.getline() iz idućeg retka 	<code>iostream</code>
<code>cin.getline(polje</code> <code>charova, dužina)</code>	učitava sve znakove do kraja retka ili do unesene dužine što je 2. parametar	<code>iostream</code>
<code>cin.ignore()</code>	zanemaruje preostali sadržaj (ugl. enter nakon cin-a)	<code>iostream</code>
<code>strcpy(prvi, drugi)</code>	string copy - prepisuje vrijednosti drugog znakovnog niza u prvi	<code>cstring</code>
<code>strcmp(prvi, drugi)</code>	string compare - ako je prvi < drugoga, vraća negativno, ako su jednaki 0, inače pozitivno	<code>cstring</code>
<code>struct naziv {elementi}</code>	definira korisnički tip podatka s nazivom <i>naziv</i> ; stvara slogan naziv	
<code>typedef naziv;</code>	definira korisnički tip podatka s nazivom <i>naziv</i> ; koristimo pri vezanim listama	
<code>return;</code>	Izlaz iz funkcije, za funkcije koje nisu tipa void treba još dati i vrijednost s kojom funkcija izlazi.	

16. Literatura

Lovrenčić, A., & Konecki, M. (2018). Programiranje u 14 lekcija. Zagreb: Element

Kusalić, D. (2010). Napredno programiranje i algoritmi u C-u i C++-u. Zagreb: Graphis.