Funkcije. Varijabilni parametri. Polja kao parametri. Rekurzije.

Ciljevi:

- upoznati se s pojmom varijabilnih parametara
- proučiti uporabu polja kao parametra funkcije
- upoznati se sa pojmom rekurzije
- naučiti primijeniti navedene pojmove u radu s funkcijama

Pregled lekcije

Varijabilni parametri

Uzmimo primjer u kojem korisnik unosi 2-5 prirodnih brojeva te za iste treba izračunati njihovu najveću zajedničku mjeru. Rješenje treba uraditi bez korištenja polja ili bilo kojeg drugog mehanizma agregacije.

Primjer 1:
ULAZ: Unos od 2 do 5 prirodnih brojeva
IZLAZ: Ispisati najveću zajedničku mjeru (NZM)

S algoritmom traženje najveće zajedničke mjere smo se već susretali u zadacima jedne od prethodnih lekcija. Za traženje NZM koristit ćemo Euklidov algoritam koji ćemo realizirati pomoću funkcije.

Euklidov algoritam:

1. Ako je A<B zamijeni im mjesta
2. Sve dok je B != 0
3. pom = A%B
4. A = B
5. B = pom
6. vrati A

U našem primjeru moramo u najgorem slučaju napraviti NZM 5 broja. Kako izračunati NZM za više od dva broja? NZM u slučaju kada imamo više od dva broja računamo tako da prvo izračunamo NZM prva dva broja, nakon toga NZM mjere prvih dvaju brojeva i trećeg broja, itd.

Navedeni problem možemo riješiti uporabom funkcije koja će primati 2, 3, 4 ili 5 parametara. Nažalost takvo što nije moguće, no moguće je simulirati ovakvu funkcije na način da napravimo funkciju u kojoj će se parametri koji nedostaju popuniti predefiniranim (default) vrijednostima. U slučaju našeg zadatka, sve predefinirane vrijednosti ćemo postaviti na 1 jer je NZM broja A i 1 sam broj A i predefinirane vrijednosti ne utječu na rezultata.

Pojednostavljeno, ukoliko ne proslijedimo parametar za njegovu vrijednost će se uzeti 1 tj. kod poziva funkcije predefinirani parametri se mogu izostaviti i bit će popunjeni predviđenom vrijednošću tj. 1 u našem slučaju. Parametri s predviđenom vrijednošću moraju biti na kraju liste parametara funkcije.

Dakle, koraci dolaženja do NZM 2-5 brojeva su:

1. R = M(A,B)
2. Ako je C > 1 onda R = M(R,C)
3. Ako je D > 1 onda R = M(R,D)
4. Ako je E > 1 onda R = M(R,E)
5. vrati R

U glavnom program ćemo od korisnika tražiti unos brojeva i ovisno o tome koliko brojeva korisnik unese, proslijedit ćemo odgovarajući broj parametara funkciji za izračun NZM.

1. Unijeti N
2. Ako je N = 5
3. Učitati A,B,C,D,E
4. ispisati M(A,B,C,D,E)
5. Ako je N = 4
6. Učitati A,B,C,D
7. ispisati M(A,B,C,D)
8. Ako je N = 3
9. Učitati A,B,C
10. ispisati M(A,B,C)
11. Ako je N = 2
12. Učitati A,B

ispisati M(A,B)

Konačno programsko rješenje našeg zadatka glasi:

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

int M(unsigned int A, unsigned int B) {
    if (A<B) {
       int pom = A;
       A = B;
       B = pom;
    }
    while (B != 0) {
          int pom = A%B;
          A = B;
          B = pom;
    }
    return A;
}
```

```
int MM(int A, int B, int C=1, int D=1, int E=1) {
    int R=M(A,B);
    if (C>1) R = M(R,C);
    if (D>1) R = M(R,D);
    if (E>1) R = M(R,E);
    return R;
}

int Ucitaj() {
    int b;
    do {
       cout << "Upisite prirodan broj: ";
       cin >> b;
    } while (b<1);
    return b;
}

int main () {
    int N, A, B, C, D, E;
    do {
       cout << "Koliko brojeva zelite?";
       cin >> N;
    } while (N<2 || N>5);
    switch (N) {
           case 5: E = Ucitaj();
           case 4: D = Ucitaj();
           case 3: C = Ucitaj();
           case 2: B = Ucitaj();
                   A = Ucitaj();
    }
switch (N) {
           case 5: cout << MM(A,B,C,D,E);
                   break;
           case 4: cout << MM(A,B,C,D);
                   break;
           case 3: cout << MM(A,B,C);
                   break;
           case 2: cout << MM(A,B);
    }
    cout << endl;
    system("PAUSE");
    return 0;
}
```

Polja kao parametri, izmjena parametara funkcije

Primjer 2:

U sljedećem zadatku ćemo pokazati kako funkciji proslijediti polje kao parametar i kako napraviti da se parametri funkcije mijenjaju, a da te promjene budu vidljive izvan funkcije (bit će pokazana dva načina).

ULAZ: Polje točaka u ravnini. Svaka je točka određena dvjema koordinatama, koje su decimalni brojevi.
IZLAZ: Dvije točke koje su međusobno najmanje udaljene i njihovu udaljenost.

Udaljenost točaka $T_1 = (x_1, y_1)$ i $T_2 = (x_2, y_2)$ izračunava se po formuli
U polju nije potrebno izračunavati udaljenost svake točke T_i i T_j, jer je $d(T_i, T_j) = d(T_j, T_i)$
Potrebno je za svaki i provjeriti udaljenost točke T_i sa točkama T_j za koje je $j>i$.

Problem jest kako inicijalizirati udaljenost na početku petlje? Kako bismo održali uniformnost rada programa, najbolje je da kao inicijalnu vrijednost uzmemo udaljenost posljednje dvije točke u nizu, te da petlju započnemo usporedbom točaka s točkom koja je treća straga.

Koraci rješenja našeg problema su:

1. D = d(T_{N-1}, T_{N-2})
2. T1 = N-2, T2=N-1
3. Za I = N-3..0 radi
4. Za J=I+1..N-1 radi
5. Ako je d(T_I, T_J) < D onda
6. D = d(T_I, T_J)
7. T1 = I
8. T2 = J
9. Vrati D, T1, T2

Ova funkcija mora vratiti 3 vrijednosti: udaljenost i točke koje su međusobno najbliže. Mogli bismo sve te tri vrijednosti "upakirati" u slog i kao povratni tip iz funkcije definirati taj slog. No moguće je te vrijednosti definirati kao parametre čije su promjene vidljive i nakon povratka iz funkcije.

Za početak ćemo polje koje sadrži točke koje se razmatraju definirati globalno. To je polje točaka, a svaka će točka biti definirana kao slog koji sadrži 2 vrijednosti tipa double (koordinate) a kako bismo pojednostavnili glavnu funkciju, napravit ćemo još jednu funkciju za izračun udaljenosti dviju točaka. Da bismo postigli da C++ u parametre ne vrati vrijednosti sa stoga koristit ćemo pokazivače kao parametre funkcije a ne varijable. Time postižemo da će C++ na programski stog zapisati i s njega vratiti vrijednost pokazivača, a ne same varijable, tako da će promjena vrijednosti varijable "preživjeti" povratak iz funkcije.

Problem koji sada nastaje je da u ćemo u glavnom programu deklarirati varijable elementarnih tipova, a u funkciju trebamo prenijeti njihove adrese. Da bismo to uraditi koristiti ćemo operator & koji je inverzan operatoru * i naziva se operator adrese tj. on vraća adresu varijable koja se nalazi iza njega.

Dakle, programski kod našeg rješenja bi glasio:

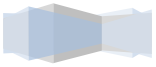
```
#include <iostream>
#include <cmath>
using namespace std;

struct tocka {
	double x,y;
};

tocka T[1000];
```

```
double D(tocka T1, tocka T2) {
       return sqrt(pow(T1.x-T2.x,2) +
                   pow(T1.y-T2.y,2));
}

void Udaljenost (int n, int *i, int *j, double *d) {
     *d = D(T[n-2], T[n-1]);
     *i = n-2;
     *j = n-1;
     for (int I=n-3; I>=0; I--)
         for (int J=I+1; J<n; J++) {
             double d1 = D(T[I], T[J]);
             if (d1 < *d) {
                *d = d1;
                *i=I;
                *j=J;
             }
         }
}

int main () {
    int N, i,j;
    double d;
    cout << "Koliko tocaka zelite unijeti? ";
    do {
       cin >> N;
    } while (N<1);
    for (int i=0; i<N; i++) {
        cout << "Unesite " << i << "-tu tocku: ";
        cin >> T[i].x >> T[i].y;
    }
    Udaljenost(N,&i,&j,&d);
    cout << "Najblize su tocke T_" << i
         << " i T_" << j << ".\n";
    cout << "Njihova udaljenost iznosi "
         << d << endl;
    system("PAUSE");
    return 0;
}
```

Da bi se riješio problem prosljeđivanja varijabli funkcijama spomenut u prethodnom odlomku može se koristiti još jedan mehanizam C++-a koji je napravljen baš za tu svrhu ali ne mnogo rjeđe koristi. Taj mehanizam se naziva referenca. Referenca služi kako bi se definirala lokalna varijabla koja se pri pozivu funkcije ne stavlja na programski stog.

Tako da bi naše rješenje primjenom referenci izgledalo ovako:

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

struct tocka {
       double x,y;
};
```

```
tocka T[1000];

double D(tocka T1, tocka T2) {
       return sqrt(pow(T1.x-T2.x,2) +
                   pow(T1.y-T2.y,2));
}

void Udaljenost (int n, int& i, int& j, double& d) {
     d = D(T[n-2], T[n-1]);
     i = n-2;
     j = n-1;
     for (int I=n-3; I>=0; I--)
         for (int J=I+1; J<n; J++) {
             double d1 = D(T[I], T[J]);
             if (d1 < d) {
                d = d1;
                i=I;
                j=J;
             }
         }
}

int main () {
    int N, i,j;
    double d;
    cout << "Koliko tocaka zelite unijeti? ";
    do {
       cin >> N;
    } while (N<1);
    for (int i=0; i<N; i++) {
        cout << "Unesite " << i << "-tu tocku: ";
        cin >> T[i].x >> T[i].y;
    }
    Udaljenost(N,i,j,d);
    cout << "Najblize su tocke T_" << i
         << " i T_" << j << ".\n";
    cout << "Njihova udaljenost iznosi "
         << d << endl;
    system("PAUSE");
    return 0;
}
```

Razlike su u parametrima i pozivu funkcije Udaljenost.

Već smo rekli da su reference napravljene isključivo za rješavanje navedenog problema i da se u pravilu rijetko koriste. Koja je razlika između referenci i pokazivača?

Pokazivači	Reference
Pokazivači omogućuju dinamičku alokaciju memorijskog prostora	Reference samo pokazuju na statički alociranu varijablu
Vrijednost pokazivača se može mijenjati	Vrijednost reference je fiksna
Pokazivači u C++ su "nesigurni"	Reference nisu varijable same po se bi i preko njih se ne može pristupati nealociranom memorijskom prostoru

Još jedan problem našeg rješenja je taj što smo polje alocirali statički i njegova dimenzija ne reflektira stvarni broj podataka. To smo učinili zato što globalno polje ne možemo alocirati dinamički. Ako prijeđemo na dinamičku alokaciju polja, bit će potrebno polje prenijeti kao argument funkcije.

Kada bi se polje prenosilo u funkciju kao ostale varijable, zapisujući ga na programski stog i vraćajući vrijednosti s programskog stoga nakon povratka iz funkcije, to bi bilo krajnje neefikasno. Zbog toga se u funkciju prenosi samo pokazivač na polje. To pak uzrokuje da će sve promjene na elementima polja ostati promijenjene i nakon povratka iz funkcije.

Ako izmijenimo naše rješenje da kao parametar prima polje tada će ono glasiti ovako:

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

struct tocka {
       double x,y;
};

double D(tocka T1, tocka T2) {
       return sqrt(pow(T1.x-T2.x,2) +
                   pow(T1.y-T2.y,2));
}

void Udaljenost (int n, tocka T[], int& i, int& j, double& d) {
     d = D(T[n-2], T[n-1]);
     i = n-2;
     j = n-1;
     for (int I=n-3; I>=0; I--)
         for (int J=I+1; J<n; J++) {
             double d1 = D(T[I], T[J]);
             if (d1 < d) {
                d = d1;
                i=I;
                j=J;
             }
         }
}
```

```
int main () {
    int N, i,j;
    double d;
    cout << "Koliko tocaka zelite unijeti? ";
    do {
       cin >> N;
    } while (N<1);
tocka *T = new tocka [N];
    for (int i=0; i<N; i++) {
        cout << "Unesite " << i << "-tu tocku: ";
        cin >> T[i].x >> T[i].y;
    }
    Udaljenost(N,T,i,j,d);
    cout << "Najblize su tocke T_" << i
         << " i T_" << j << ".\n";
    cout << "Njihova udaljenost iznosi "
         << d << endl;
    system("PAUSE");
    return 0;
}
```

Rekurzija

Uzmimo kao primjer sljedeći zadatak.

Primjer 3:

ULAZ: Broj N

IZLAZ: Zadani su 3 stupa i N kolutova različitih veličina. Inicijalno su svi kolutovi postavljeni na prvi stup i to tako da se svaki kolut nalazi iznad većeg koluta. Potrebno je prebaciti sve kolutove na treći stup uz sljedeća ograničenja:

1. Odjednom se može prebacivati samo jedan kolut
2. Ni u kojem trenutku se ni jedan kolut ne smije naći iznad manjeg koluta.

Ovaj je problem poznat kao problem tornjeva Hanoja

Potrebno je riješiti ovaj problem za općeniti N. To možemo učiniti jednostavnom indukcijom. Prvo što trebamo ustanoviti je da znamo kako prebaciti 1 kolut s prvog na treći stup. Pretpostavljamo da znamo prebaciti N-1 kolut s prvog na treći stup. S tim u vidu kažemo da se N kolutova može prebaciti s prvog na treći stup na sljedeći način:

1. Prebaci *N*-1 kolut s prvog na drugi stup
2. Prebaci najveći kolut s prvog na treći stup
3. Prebaci *N*-1 kolut s drugog na treći stup

Programski stog nam omogućuje još jednu vrlo korisnu stvar – rekurziju, koja nam može pomoću u rješavanju našeg problema. Rekurzija je mehanizam koji omogućuje da funkcija poziva samu sebe s promijenjenim parametrima. Kao takva, rekurzija predstavlja još jedan mehanizam koji omogućuje iteracije (petlje) u programima, ali je mnogo moćnija od bilo koje petlje koja se može naći u bilo kojem programskom jeziku. Može se reći da je rekurzija mehanizam matematičke indukcije u programskim jezicima.

Kako generirati rekurziju:

1. Generirati nerekurzivni slučaj (bazu indukcije) koji se rješava izravno. Nerekurzivni se slučaj naziva još i rubni uvjet rekurzije
2. Generirati rekurzivni slučaj (korak indukcije) koji se rješava pozivom iste funkcije s manjim ulazom.

Napomena: Treba biti siguran da će korak rekurzije u konačno mnogo poziva ulaz transformirati u nerekurzivni slučaj.

Koraci rješenja našeg problema su:

1. Funkcija Hanoi(N, poc, kraj)
2. Ako je N = 1 onda
3. Ispiši poc "->" kraj
4. inače
5. Hanoi(N-1,poc, pom)
6. Ispiši poc "->" kraj
7. Hanoi(N-1,pom, kraj)

S time da koraci 2 i 3 predstavljaju nerekurzivni slučaj (baza) a koraci 4-7 rekurzivni slučaj (korak).

Rekurzija će raditi tako da će u svakom novom pozivu generirat će se novi set lokalnih varijabli. Varijable će se nakon povratka iz funkcije vratiti na set varijabli pozivne funkcije.
Treba imati na umu da će ovo vrijediti samo za vrijednosti koje su prenesene na programski stog, dakle ne i za varijable koje su prenesene preko pokazivača i referenci, kao ni za polja.

Stupove ćemo u našem primjeru nazvati 1, 2 i 3.
U svakom pozivu funkcije imamo zadani početni i krajnji stup. Pomoćni stup određujemo na sljedeći način: pom = 6 – poc – kraj.

Dakle, programsko rješenje našeg problema glasi:

```
#include <iostream>
using namespace std;
void Hanoi(int N, char poc, char kraj) {
     if (N == 1)
        cout << (int)poc << "->"
             << (int)kraj << endl;
     else {
          Hanoi(N-1, poc,6-poc-kraj);
          cout << (int)poc << "->"
               << (int)kraj << endl;
          Hanoi(N-1, 6-poc-kraj, kraj);
     }
}

int main () {
    int N;
    do {
```

```
        cout << "N = ";
        cin >> N;
    } while (N<1);
    Hanoi(N,1,3);
    system("PAUSE");
    return 0;
}
```

Promotrimo sada problem izračunavanja binomnih koeficijenata. Kao što je poznato, za nenegativne brojeve K i N, K≤N vrijedi

$$\binom{N}{K} = \frac{N!}{K!(N-K)!}$$

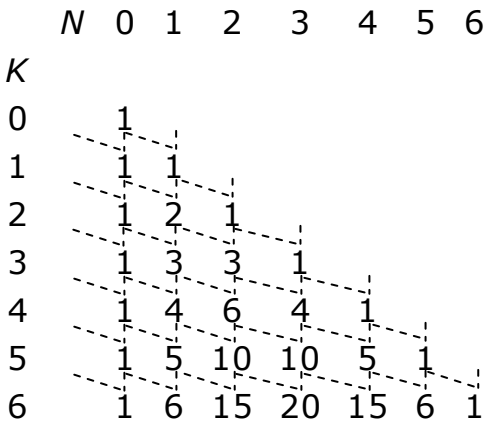
Prema ovoj formuli bismo mogli jednostavno izračunati binomni koeficijent kako slijedi:

```
#include <iostream>

using namespace std;

int Fakt(int N) {
    if (N==0)
       return 1;
    else {
         int F = 1;
         for (int i = 2; i<=N; i++)
             F *= i;
         return F;
    }
}

int main () {
    int N, K;
    do {
        cout << "N = ";
        cin >> N;
        cout << "K = ";
        cin >> K;
    } while (N<0 || K<0 || K>N);
    cout << " " << N << endl;
    cout << "(  ) = " << Fakt(N)/(Fakt(K)*Fakt(N-K)) << endl;
    cout << " " << K << endl;
    system("pause");
    return 0;
}
```

No, ovo rješenje ima ozbiljan problem – faktorijeli rastu mnogo brže nego binomni koeficijenti. Tako će on za $\binom{17}{1}$ vratiti 0, što znamo da nije točno rješenje, već je ono 17. Naime, problem je u tome što 17! prelazi interval brojeva obuhvaćenih tipom podataka `int`, pa se ni binomni koeficijent, iako je on po svom iznosu malen, ne može izračunati. Postoji, međutim jednostavna rekurzivna metoda koja ne ovisi o računanju faktorijela i moći će izračunati i znatno veće binomne koeficijente. Ona se temelji na rekurzivnoj formuli

$$\binom{N}{0} = 1$$
$$\binom{N}{N} = 1$$
$$\binom{N}{K} = \binom{N-1}{K} + \binom{N-1}{K-1}$$

Korištenjem ove formule dobijamo poznatu konstrukciju za izračunavanje binomnih koeficijenata zvanu Pascalov trokut. Za N = 0...5 Pascalov trokut izgleda kao na sljedećoj slici:



Dakle, svaki se binomni koeficijent u N-tom retku može izračunati preko dva koeficijenta u (N-1)-vom retku. Ovaj nas postupak prirodno navodi na rekurzivno rješenje problema opisano sljedećim rekurzivnim pseudokodom

ULAZ: Nenegativni brojevi N i K, K≤N

IZLAZ: $\binom{N}{K}$

1. Ako je N == K ili je K == 0
2. Vrati 1
3. inače
4. Vrati $\binom{N-1}{K} + \binom{N-1}{K-1}$

Ono što je važno provjeriti pri ovoj rekurziji jest valjanost rubnih uvjeta, tj. hoće li ova rekurzija uvijek stati. U svakom se rekurzivnom pozivu N smanjuje za 1. Tako će uvijek N dostići K u konačno mnogo koraka. S druge strane, u drugom se pribrojniku K smanjuje za 1, pa će u K koraka K postati 0. U svakom od ova dva slučaja koeficijent se izračunava izravno. Dakle, zaključujemo da je rekurzija dobro definirana.

U sljedećem je programu glavni program gotovo jednak kao i u prethodnom, ali se binomni koeficijent računa rekurzivno.

```
#include <iostream>

using namespace std;

int Koef(int N, int K) {
    if (N==K ||K==0)
       return 1;
    else
         return Koef(N-1,K) + Koef(N-1,K-1);
}

int main () {
    int N, K;
    do {
        cout << "N = ";
        cin >> N;
        cout << "K = ";
        cin >> K;
    } while (N<0 || K<0 || K>N);
    cout << " " << N << endl;
    cout << "(  ) = " << Koef(N,K) << endl;
    cout << " " << K << endl;
    system("pause");
    return 0;
}
```

Zadaci za vježbu

1. Odgovorite na pitanje što su to varijabilni parametri.
2. Odgovorite na pitanje kako se prenosi polje kao argument.
3. Odgovorite na pitanje što je to rekurzija?
4. Izradite program će pomoću rekurzije računati faktorijel bilo kojeg prirodnog broja.

Programski primjeri za laboratorijske vježbe

Zadatak 1.

Izradite program u kojem:

1. Korisnik unosi N = 1..100 i polje 1 do 100 pozitivnih cijelih brojeva. Poziva se funkcija kojoj je deklarirano polje i N argumenata i koja u slučaju da je proslijeđen 1 broj, uvećava taj broj za 2 i vraća rezultat, a inače zbraja sve proslijeđene parametre i vraća njihovu sumu kao rezultat. Rezultat se ispisuje unutar glavnog programa.

2. Korisnik unosi 3 stranice trokuta, a1, a2 i a3 u polje. Polje se potom prosljeđuje kao parametar funkciji koja računa i ispisuje površinu trokuta izračunatu pomoću Heronove formule.
   $$P = \sqrt{s \cdot (s-a) \cdot (s-b) \cdot (s-c)}$$
   gdje je
   $$s = \frac{a+b+c}{2}$$

3. Korisnik unosi broj elemenata polja. Stvara se dinamičko polje unesene dimenzije n. Korisnik potom popunjava polje sa cijelim brojevima. Ukoliko broj nije cijeli ponavlja se unos. Popunjeno polje i broj elemenata u njemu se kao parametar prosljeđuje funkciji koja rastavlja svaki pojedini broj iz polja na proste faktore i to ispisuje. Funkcija također računa i ispisuje najveću zajedničku mjeru svih brojeva u polju.

4. Program koristeći funkciju i rekurziju izračunava M-ti broj Fibonaccijevog niza F_n za n = 0, 1. Fibonaccijev niz definiran je sljedećom rekurzivnom relacijom:
   $$F(n) := \begin{cases} 0 & n = 0; \\ 1 & n = 1; \\ F_{n-1} + F_{n-2} & n > 1. \end{cases}$$

5. Problem kraljica se sastoji od postavljanja 8 kraljica na šahovskoj tabli tako da nikoje dvije kraljice ne napadaju jedna drugu. Pri tome kraljica napada drugu kraljicu ako se nalazi u istom redu, stupcu ili dijagonali šahovske table. Napraviti program koji će pomoću rekurzije rješavati problem kraljica i ispisati sva moguća rješenja.