Algoritamska struktura C++ programa

Algoritam je zapis radnji (korak po korak) koje treba obaviti da bi se u konačnom broju koraka došlo do željenog rješenja.

Tri tipa radnji su tipična za kompjuterske algoritme:

1. sekvenca naredbi
2. selekcija naredbi
3. iterativno ponavljanje sekvence (petlje)

Primjer proračuna n! (n-faktorijela):

1. Analiza zadatka: Program mora obaviti slijedeće operacije:

1. Dobaviti vrijednost od *n*.
2. Izračunati vrijednost *n*!.
3. Ispisati vrijednost od *n* i *n*!.

2. Analiza problema: Polazimo od definicije n-fatorijela

$n! = 1$ za $n = 0$

$n! = 1*2 * ..(n-2)*(n-1)*n$ za $n > 0$

Koristeći prethodnu definiciju lako je uočiti da vrijedi i slijedeće pravilo

$n! = 1$ za $n = 0$

$n! = n * (n-1)!$ za $n>0$

koje kazuje da se vrijednost od n! može izračunati iz prethodno poznate vrijednosti od $(n-1)!$.

Algoritam dorađujemo čim ih postavimo!

Algoritam za korak 2. Izračunati vrijednost *n*!.

2.1. Postavi nfact = 1;
2.2. Postavi k=0;
2.3. Dok je k < n ponavljaj
    Uvećaj vrijednost varijable k za jedan
    nfact = k * nfact;

Dorada koraka 2.

2.1 Postavi nfact = 1;
2.3 Postavi k=1; !!!!!! za k=0 i k=1: n!=1
2.3 Dok je k < n ponavljaj
    Uvećaj vrijednost varijable k za jedan
    nfact = k * nfact;

Implemantacija algoritma

```
// Proračun n!. Vrijednost od n unosi korisnik.
#include <iostream>
using namespace std;

int main()
{
   int n, k, nfact;   // deklaracija varijabli

   // korak 1
   cin >> n;

   // korak 2
   nfact = 1;                    // korak 2.1
   k = 1;                        // korak 2.2
   while ( k < n)                // korak 2.3
   {
       k = k + 1;
       nfact = k * nfact;
   }

   // korak 3
   cout << "Vrijednost" << n << "! "
        << "iznosi: " << nfact << endl;
   return 0;
}
```

Općenito while-petlja ima oblik:

```
while (izraz)
{
 niz_naredbi        ili       while (izraz) naredba
}
```

Značenje: dok je (eng. while) *izraz* u zaglavlju petlje različit od nule izvršava se *niz_naredbi* tijela petlje koje su napisane unutar vitičastih zagrada. Ako je *izraz* jednak nuli izvršenje programa se nastavlja operacijom koja je definirana naredbom koja slijedi iza tijela petlje.

U slučaju kada se u tijelu petlje navodi samo jedna naredba, tada nije nužno pisati vitičaste zagrade.

Uvijek analiziraj problem!

Ponovna analiza: Program daje nesuvisle rezultate ako je n<0 ili ako je n>13 (tada je rezultat veći od MAX_INT)

Dorada koraka 1: Dobaviti vrijednost od n.

1.1. Upozoriti korisnika da se očekuje unos broja unutar intervala [0,13]
1.2. Dobaviti otipkanu vrijednost u varijablu n
1.3 Ako je n < 0 ili n > 13 tada izvršiti sljedeće:
izvijestiti korisnika da je otkucao nedozvoljeni broj
prekinuti izvršenje programa

Za implementaciju koraka 1.3 potrebno je upoznati kako se u C jeziku zapisuje uvjetna naredba tzv. if-naredba. Njen opći oblik glasi:

```
if (izraz)
{
   niz_naredbi          ili      if (izraz) naredba;
}
```

a značenje je: ako je (eng. if) *izraz* različit od nule izvršava se *niz_naredbi* koji je omeđen vitičastim zagradama, u protivnom izvršit će se naredba koja slijedi iza if-naredbe.

If naredba i logički operator 'ili'

Implementacija koraka 1:

```
cout << "Unesite broj unutar intervala [0,13]\n";
cin >> n;

if((n < 0) || (n > 13))
{
  cout << "Otipkali ste nedozvoljenu vrijednost";
  return 1;    // forsirani izlaz iz funkcije main
}
```

n<0 i n>13 su relacijski izrazi koji čiji rezultat može biti logička vrijednost true ili false (1 ili 0)

Logički operator ili (||) djeluje prema tablici

A	B	a || b
false	false	false
false	true	true
true	false	true
true	true	true

(a || b) je TRUE ako je jedan operand TRUE

If-else naredba

Mogli smo rezonirati i ovako:

Dobavi vrijednost od n.
Ako je n >= 0 i n<=13 tada !!! logički operator "i" !!!
    Izračunj vrijednost n!.
    Ispiši vrijednost od n i n!.
inače
 Izvijesti o pogrešnom unosu
Kraj!

Upoznajmo if-else naredbu i logički operator "i"

```
if(izraz)
{
    niz_naredbi1
}
else
{
   niz_naredbi2
}
```

ili

```
if(izraz)
    naredba1;
 else
    naredba2;
```

Značenje je: ako je *izraz* različit od nule izvršava se *niz_naredbi1,* inače (eng. else) izvršava se *niz_naredbi2*. Ako *niz_naredbi* sadrži zamo jednu naredbu ne moraju se pisati vitičaste zagrade. *Izraz* se tretira kao logička vrijednost.

Logički operator 'i' (&&)

```
if((n >= 0) && (n<=13))  // logički operator "i"
{
   // Izračunj vrijednost n!.
   // Ispiši vrijednost od n i n!.
}
else
{
   //Izvijesti o pogrešnom unosu
}
```

Logički operator "i" (&&) djeluje prema tablici

a	b	a && b
false	false	False
false	true	false
true	false	false
true	true	true

(a && b) je TRUE ako su oba operanda jednaka TRUE

Funkcije - poopćenje primjene algoritama

Proračun *n*! je univerzalni problem, stoga ćemo sada definirati funkciju imena factorial kojom se obavlja taj proračun. Prototip te funkcije može biti oblika:

```
int factorial(int n);
```

Kao argument funkcija će koristiti vrijednost tipa int. Primjena funkcije u izrazima rezultirat će vrijednošću tipa int koji predstavlja vrijednost n-faktorijela. Definicija i primjena funkcije zapisani su u programu fact4.cpp.

```
/* factorial: funkcija za proračun n faktorijela
 * PRE:  funkcija daje ispravan rezultat ako je
 * vrijednost ulaznog parametra unutar intervala
 * 0 <= n <= 13
 * POST: rezultat funkcije je vrijednost od n!
 ************************************************/

int factorial(int n)
{
   int k = 1, nfact = 1;
   while (k < n) {
      k = k + 1;
      nfact = k * nfact;
   }
   return nfact;
}
```

```
int main()
{
   int n;
   cout << "Unesite broj unutar intervala [0,13]\n";
   cin >> n;

   if((n < 0) || (n > 13))
         cout << "Otipkali ste nedozvoljenu vrijednost";
   else
        cout << "Vrijednost " << n << "! "
            << "iznosi: " << factorial(n) << endl;

   return 0;
}
```

Strukturalno programiranje

REALIZIBILNOST ALGORITAMA

Svaki se algoritam može realizirati programskim jezikom koji ima tri tipa naredbi:

1. sekvencu naredbi — a = b
2. selekciju naredbi — if-else
3. iterativno ponavljanje sekvence (petlje) — while petlju

METODA RAZVOJA ALGORITAMA - STEPWISE REFINEMENT

1. Formulirati problem na način da bude potpuno jasno **što program treba obaviti**.
2. Formulirati temeljni tijek algoritamskog rješenja, običnim govornim jezikom.
3. Izdvojiti pogodnu manju cjelinu i razložiti je na detaljnije programske zahtjeve.
4. Ponavljati korak (3) dok se ne dobiju programski zahtjevi koji su toliko jednostavni da se mogu realizirati programskim jezikom.
5. Odabrati neki od programskih zahtjeva i realizirati ga programskim jezikom (ili pseudojezikom). Pri tome treba odrediti potrebne struktura podataka.
6. Sustavno ponavljati korak (5) i pri tome povećavati razinu dorade programskih rješenja.

Karakteristike "dobrog" programa:

- robusnost – daje suvisli rezultat za bilo koji ulaz
- čitljivost – lako ga je razumjeti i modificirati
- efikasnost – brzo se izvršava i upotrebljava malo memorije
- općenitost – rješava problem za širi skup vrijednosti

Funkcije s predodređenim parametrima

U C++ jeziku (ne i u C jeziku) može se u deklaraciji funkcije navesti i vrijednost nekog parametra.

```
void fun(int a, int b = 2);
```

Takovi parametri se nazivaju predodređeni (default) parametri funkcije. Moguće je izvršiti poziv prethodne funkcije bez drugog argumenta;

```
fun(a);
```

tada kompajler sam generira drugi argument iz prethodne deklaracije, tj.
stvarno se vrši poziv s dva argumenta `fun(a, 2);`

Pravila za korištenje predodređenih parametara

- Može biti više predodređenih parametara
- Predodređeni parametri se zapisuju na kraju liste parametara funkcije. Primjerice, ako funkcija ima dva predodređena parametra od ukupno tri parametra, tad dva predodređena parametra moraju biti posljednji s desna u listi parametara.
- Pri pozivu funkcije, moraju biti zadani argumenti s lijeva. Primjerice, samo prvi argument s lijeva ili prvi i drugi argument s lijeva, ili sva tri argumenta.
- Svi predodređeni parametri se moraju prenositi po vrijednosti.

Za funkciju:

```
void bozo(int a, int b = 2, int c = 5);
```

mogući su pozivi:

```
bozo(7, 6, 3); // OK. Poziv bez predodređenih vrijednosti
bozo(7, 6);    // OK. Isto kao i bozo(7, 6, 5);
bozo(7);       // OK. Isto kao i bozo(7, 2, 5);
bozo();        // Greška. Parametar a mora biti prisutan.
bozo(7, , 3);  // Greška.
```

Primjer: Proračun funkcije e^x

Zadatak:

napisati funkciju kojom se približno određuje vrijednost funkcije e^x (e = 2. 718282). Rezultat usporediti s vrijednošću koja se dobije pomoću standardne funkcije exp(x), kojoj je prototip – `double exp(double x)` - deklariran u datoteci `math.h`.

Metod:

Koristeći razvoj u red: $e^x = 1 + x/1! + x^2/2! + x^3/3! + ..$

zbrajati članovi reda, za dati x, sve dok razlika od prethodnog rezultata ne bude manja od zadane preciznosti eps. Primjerice, za x = 1.0, i eps = 0.0001 trebat će zbrojiti 10 članova reda.

```
double my_exp(double x, double epsilon = 0.00001)

int main( void)
{
    double x, ex;
    cout << " Unesi x :\n";   cin >> x;

    ex = my_exp(x);  // nema drugog argumenta

    cout << " e^ << x << "=" << ex
         << " (tocno: " << exp(x) << ")" << endl;
    return 0;
}
```

```
double my_exp(double x, double epsilon)
{
   int i = 1;
   double pribroj = 1.0;
   double ex = 1.0, preth_ex = 0.0;

   while (fabs( ex - preth_ex) > epsilon)
   {
         preth_ex = ex;
         pribroj = pribroj * x / i;
         ex = ex + pribroj;
         i = i + 1;
   }
   return ex;
}
```

Izvršenjem programa dobije su rezultati:

```
c:>ex
Unesi x i preciznost eps: 1 .00001
e^1.000000 = 2.718282; (tocno: 2.718282)

c:>ex.exe
Enter x and the eps: 2 .0001
e^2.000000 = 7.389047; (tocno: 7.389056)
```