Pagrindiniai pavyzdžiai ir algoritmai:

C++ kalbos funkcija	Atitikmuo matematikoje	Paaiškinimas
abs(x)	Modulis |x|	x - realus skaičius *Pvz.: abs(-3.5)=3.5*
fabs(x)	Modulis |x|	x - sveikas skaičius *Pvz.: abs(-3)=3*
pow(a, x)	Kėlimas laipsniu a^x	a ir x reikalavimai kaip matematikoje Pvz.: pow(3,2)=9
sqrt(x)	Šaknis	x >= 0 Pvz.: sqrt(9)=3
ceil(x)	Apvalinimas su perteklium	Pvz.: ceil(3.4)=4 ceil(3.6)=4
round(x)	Apvalinimas	Pvz.: round(3.4)=3 round(3.6)=4
floor(x)	Apvalinimas su trūkumu	Pvz.: floor(3.6)=3 floor(3.6)=3

S - pradinis skaičius, S1, S2 ir t.t. - skaičiaus S skaitmenys.

Dviženklis skaičius:
S1 = S /10;
S2 = S % 10;
Skaičiaus sudarymas iš skaitmenų:
S = S1 * 10 + S2;

Triženklis skaičius:
S1 = S /100;
S2 = (S /10) % 10;
S3 = S % 10;
Skaičiaus sudarymas iš skaitmenų:
S = S1 * 100 + S2 * 10 + S3;

Keturženklis skaičius:
S1 = S /1000;
S2 = (S /100) % 10;
S3 = (S /10) % 10;
S4 = S % 10;
Skaičiaus sudarymas iš skaitmenų:
S = S1 * 1000 + S2 * 100 + S3 * 10 + S4;

Penkiaženklis skaičius:
S1 = S /10000;
S2 = (S /1000) % 10;
S3 = (S /100) % 10;
S4 = (S /10) % 10;
S5 = S % 10;
Skaičiaus sudarymas iš skaitmenų:
S = S1 * 10000 + S2 * 1000 + S3 * 100 + S4 * 10 + S5;

Vidurkio skaičiavimo algoritmas

```
//Skaičių vidurkis, kai žinomas skaičių kiekis
...
int suma, sk, vid; //suma ir dėmuo, vidurkis
int n;       //skaičių kiekis
int i;       //ciklo kintamasis
cout << "Kiek bus įvedama skaičių ";
cin >> n;
   suma = 0;
for (i = 1; i <= n; i = i  + 1)
   {
   cout <<"Koks bus "<< i <<" skaičius? ";
   cin >> sk;
   suma = suma + sk; //arba suma += sk;
   }
vid = suma / n;
cout <<"Įvestų skaičių vidurkis = "<< vid <<endl;
...
```

Sandaugos skaičiavimo algoritmas

```
//Skaičių sandauga
...
int sand, sk; //sandauga ir dėmuo
int n;       //skaičių kiekis
int i;       //ciklo kintamasis
cout << "Kiek bus įvedama skaičių ";
cin >> n;
   sand = 1;
for (i = 1; i <= n; i = i  + 1)
   {
   cout <<"Koks bus "<< i <<" skaičius? ";
   cin >> sk;
   sand = sand * sk; //arba sand *= sk;
   }
cout <<"Įvestų skaičių sandauga = "<< sand <<endl;
...
```

4.8. Didžiausios (mažiausios) reikšmės paieška

Tai tradiciniai programavimo uždaviniai. Populiariausias jų sprendimo būdas yra toks. Ieškant didžiausios reikšmės aprašomi du kintamieji: įvedamai *x* ir didžiausiai *d* reikšmėms atmintyje laikyti. Įvedant pirmąją reikšmę, daroma prielaida, kad ši yra didžiausia:

```
d = x;
```

Po to paeiliui skaitomos kitos reikšmės ir lyginamos su *d*. Jei randama didesnė, kintamojo *d* reikšmė keičiama nauja:

```
if (x > d) d = x;
```

Taip patikrinus visą įvedamą duomenų srautą, kintamojo *d* reikšmė bus didžiausia įvesta reikšmė.

Analogiškai ieškoma mažiausios reikšmės. Skiriasi tik palyginimo sąlyga:

```
if (x < d) d = x;
```

Algoritmas, užrašytas C++ programavimo kalba:

```
...
int n, x, d, i;
cout << "Kiek bus skaičių: ";
cin >> n;
cout << "Koks 1 skaičius: ";
cin >> d;
for (i = 2; i <= n; i = i + 1) {
  cout << "Koks " << i << " skaičius: ";
  cin >> x;
  if (x > d) d = x;
}
cout << "Didžiausias skaičius: " << d;
...
```

Jei pradinė max (min) reikšmė *(čia* d*)* neįvedama, tai būtina ją priskirti: ieškant min – pakankamai didelę, ieškant max – pakankamai mažą (pvz min=1000; max=-1000)

Sumos apskaičiavimo algoritmas

Sumos apskaičiavimo algoritmas	
Suma = 0	
Kartoti, kol bus naujų dėmenų	
	Gauti dėmenį
	Suma = Suma + dėmuo
Grąžinti Suma	

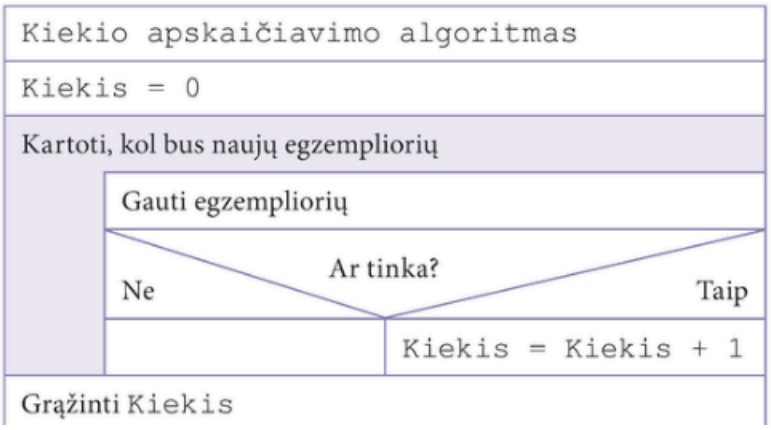
Sandaugos apskaičiavimo algoritmas

Sándaugos apskaičiãvimo algorìtmas mažai skiriasi nuo sumos apskaičiavimo algoritmo: iš pradžių sandaugai priskiriamas vienetas, o, atliekant ciklo veiksmus, sandauga daugi-nama iš naujo daugiklio:

Sandaugos apskaičiavimo algoritmas	
Sandauga = 1	
Kartoti, kol bus naujų daugiklių	
	Gauti daugiklį
	Sandauga = Sandauga * daugiklis
Grąžinti Sandauga	

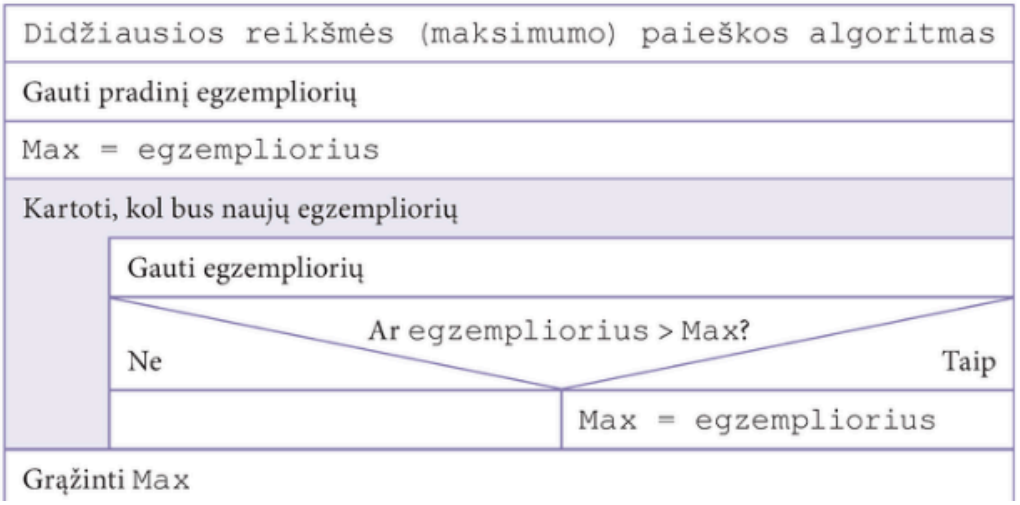
Kiekio apskaičiavimo algoritmas

Kiẽkio apskaičiãvimo algorìtmas nuo sumos apskaičiavimo algoritmo skiriasi tik tuo, kad jame sumuojami vienetai, jeigu tenkinama egzemplioriaus atrankos sąlyga:



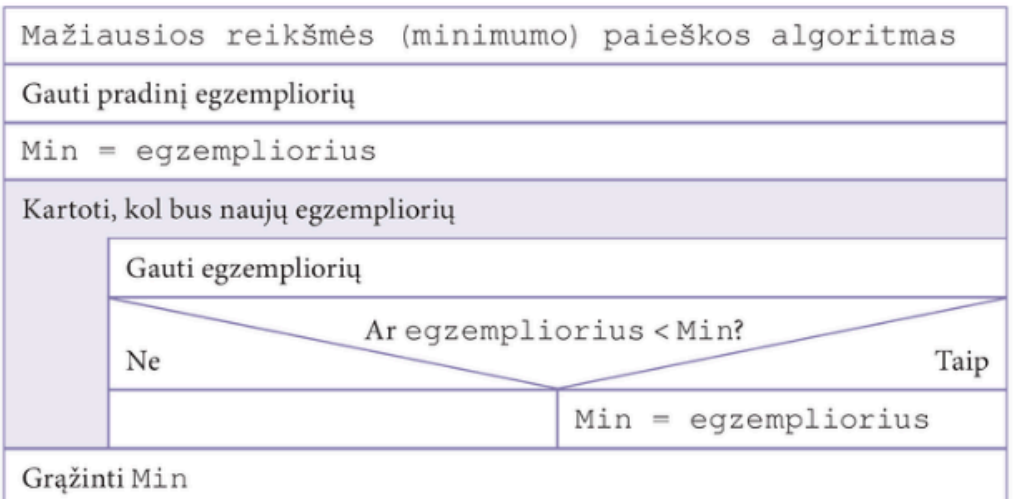
Didžiausios reikšmės (maksimumo) paieškos algoritmas

Didžiáusios reikšmẽs paieškõs algorìtmas remiasi prielaida, kad didžiausias yra pradinis egzempliorius. Toliau vykdant ciklą bandoma pagerinti rezultatą – jei randamas tinkamesnis egzempliorius, jis įsimenamas vietoj anksčiau įsimintojo.



Mažiausios reikšmės (minimumo) paieškos algoritmas

Mažiáusios reikšmẽs paieškõs algorìtmas nuo didžiausios reikšmės paieškos algoritmo skiriasi tik egzemplioriaus parinkimo sąlyga:



Aritmetinio vidurkio skaičiavimas

Aritmetinis vidurkis skaičiuojamas dviem etapais:

1) apskaičiuojama skaičių suma;
2) gauta suma padalijama iš skaičių kiekio.

Sumos skaičiavimo algoritmą jau išnagrinėjome. Reikia atkreipti dėmesį, kad jeigu iš anksto nežinoma, kiek skaičių sumuojama, tai sumuojant reikia kartu skaičiuoti, kiek tokių skaičių buvo.

Antrasis etapas – tai dviejų skaičių (gautos sumos ir skaičių kiekio) dalyba:

```
vidurkis = suma / skaičiųKiekis.
```

Žinome, kad dalyba iš nulio negalima. Todėl, prieš atliekant dalybos veiksmą (pvz., kai reikia apskaičiuoti teigiamų skaičių aritmetinį vidurkį), būtina įsitikinti, kad galima dalyti (kad įvesti ne vien neigiami skaičiai ir nuliai).

Simboliai ir skaičiai

1 pavyzdys. Tekstiniame faile yra įrašytas posakis `Errare humanum est` („Klysti yra žmogiška“). Posakį sėkmingai perskaitytume atlikę tokius veiksmus:

```
...
const int ilgis = 256;
char posakis[ilgis];
ifstream fd("duom.txt");
fd.getline(posakis, ilgis, '\n'); //skaitome posakį
...
```

Jei būtume skaitę tokiu sakiniu: `fd>>posakis;` simbolių eilutė būtų perskaityta iki pirmojo tarpo.

2 pavyzdys.

```
const int ilgis = 41;
...
duom.get(posakis, ilgis); //skaitome posakį (40 simbolių)
```

3 pavyzdys. Įvedami vardas ir pavardė. Programos rezultatas – inicialai.

```
char v, p;
cout << "Iveskite savo varda ir pavarde:";
v = cin.get();          //skaitome vardo inicialą
cin.ignore(20, ' ');    //praleidžiame iki tarpo arba 20 simbolių
cin >> ws;              // praleidžiame, jei dar yra tarpų
p = cin.get();          // skaitome pavardės inicialą
cout << "Jūsų inicialai  " << v << ". " << p << ". ";
```

Pagrindinės paprogramės (funkcijos):

Masyvo elementų suma

```
int  Suma (int A[], int n)
{
   int sum = 0;
   for (int i =  0; i < n; i++)
     sum += A[i];

  return sum;
}
```

Elemento paieška

```
bool paieska1 (int A[], int n, int r)
{
   bool taip = false;
   for (int i = 0; i < n; i++)
      if (A[i] == r ) taip = true;

   return taip;
}
```

vietoje sum+= galima naudoti įprastą sum=sum+A[i]

`bool` C++ kalboje yra **loginis duomenų tipas**, kuris gali turėti tik dvi reikšmes: `true` – tiesa ir `false` – netiesa. Jis dažniausiai naudojamas sąlygoms, palyginimams ir ciklams.

Masyvo elementų kiekio pagal duotą požymį skaičiavimas

```
int Kiekis  (int A[], int n)
{
   int kiek = 0;
   for (int i = 1 ; i <= n; i ++)
      if (A[i] == 30) kiek ++; //požymis: elementas =30
   return kiek;
}
```

vietoje kiek++ galima naudoti įprastą kiek=kiek+1

---

1 pavyzdys. Elemento pašalinimas iš vienmačio masyvo. Funkcijai teikiami pradiniai duomenys – masyvas, jo elementų skaičius ir šalinamo elemento matematinis indeksas. Pašalinus elementą, masyvo elementų skaičius sumažės vienetu, todėl šis duomuo po funkcijos darbo turės būti pakeistas – jį teks perduoti adresu. Be to, funkcijoje patikrinsim, ar šalinamo elemento indeksas yra leistinas: jo reikšmė turi būti nuo 1 iki pradinio elementų skaičiaus. Kadangi funkcijoje tam atvejui numatysim spausdinti atitinkamą pranešimą ir nutrauksim programos darbą, reikės antraštinių failų *iostream* ir *cstdlib*. Jei kviečiančioji programa ir ši funkcija bus įrašyta į vieną pradinį failą (kaip visos 5-o skyriaus programos), šiuos antraštinius failus reiks įrašyti prieš startinę funkciją. Jei visos funkcijos bus saugomos atskiruose projekto pradiniuose failuose, šių antraštinių failų prireiks ir prieš pačios funkcijos tekstą.

```
void delete_element( double x[ ], int& n, int k ){
   //
   //Argumentai:     x - pradinis masyvas
   //                n – masyvo elementų skaičius
   //                k – šalinamo elemento matematinis indeksas
   //Rezultatai:     darbo metu pakeičiamos x ir n reikšmės
   //
   //Funkcijai reikia antraštinių failų iostream ir cstdlib
   //
   if( k < 1 || k > n ) {
      cout<<"Netinkama elemento indekso reiksme  "<<k<<endl;
      exit( 1 );  // baigti darbą
   }
   //
   for( int i = k-1; i < n-1; i++ )
      x[ i ] = x[ i+1 ];
   //
   n--;
   //
}
```

2 pavyzdys. Duotos reikšmės įterpimas po nurodyto masyvo elemento. Parašysim du funkcijos variantus. Pirmu atveju reikiamoje masyvo vietoje įterpiama reikšmė išsaugant ten buvusį elementą tarpinėje ląstelėje, o toliau po vieną elementą perstumiame link masyvo galo. Antru atveju patraukdami per elementą link masyvo galo nukopijuojame visus galinius masyvo elementus, o atsiradusioje masyvo vietoje įterpiam reikšmę.

```
void insert_element1( double x[ ], int& n, int k, double r ){
   //
   //Argumentai:     x - pradinis masyvas; jam turi būti paskirtos n+1 ląstelės
   //                n – masyvo elementų skaičius
   //                k – elemento, po kurio įterpiama reikšmė, matematinis indeksas
   //                r – įterpiama reikšmė
   //Rezultatai:     darbo metu pakeičiamos x ir n reikšmės
   //
   //Funkcijai reikia antraštinių failų iostream ir cstdlib
   //
   if( k < 0 || k > n ) {
      cout<<"Netinkama elemento, po kurio iterpiama, indekso reiksme "<<k<<endl;
      exit( 1 );  // baigti darbą
   }
   //
   double t1, t2;      // tarpinės ląstelės duomenims laikinai saugoti:
   t1 = x[ k ];        // ...išsaugosim elementą, į kurio vietą turim įdėti r
   x[ k ] = r;         // ... į jo vietą įdėsim r
   //
   for( int i = k+1; i < n+1; i++ ) { // elementų masyve padaugės iki n+1
      t2 = x[ i ];     // laikinai išsaugosim elementą, vietoj kurio įdėsim ankstesnį
      x[ i ] = t1;     // ...elementą
      t1 = t2;         // kitam ciklo žingsniui turim išsaugoti kitą elementą
   }
   //
   n++;
}
```


2 variantas

```
void insert_element2( double x[ ], int& n, int k, double r ){
   //
   //Argumentai:     x - pradinis masyvas; jam turi būti paskirtos n+1 ląstelės
   //                n – masyvo elementų skaičius
   //                k – elemento, po kurio įterpiama reikšmė, matematinis indeksas
   //                r – įterpiama reikšmė
   //Rezultatai:     darbo metu pakeičiamos x ir n reikšmės

   //Funkcijai reikia antraštinių failų iostream ir cstdlib
   //
   if( k < 0 || k > n ) {
      cout<<"Netinkama elemento, po kurio iterpiama, indekso reiksme "<<k<<endl;
      exit( 1 );  // baigti darbą
   }
   //
   for( int i = n; i > k; i-- ) // galinius elementus nuo n iki k
      x[ i ] = x[i-1];       // nukopijuosim į tolesnių elementų vietas
   x[ k ] = r;               // į reikiamą vietą įdedam reikšmę
   //
   n++;
}
```

3 pavyzdys. Didžiausio elemento masyve ir jo indekso paieška.

```
void max_element( double x[ ], int n, double& max, int& k ){
   //
   //Argumentai:      x - pradinis masyvas
   //                 n – masyvo elementų skaičius
   //Rezultatai:      max – didžiausias elementas
   //                 k – didžiausio elemento matematinis indeksas
   //
   //
   max = x[ 0 ];      // prielaida: tarkim, didžiausias yra pirmas elementas
   k = 1;             // tada jo matematinis indeksas yra 1
   //
   for( int i = 1; i < n; i++ ) {
      if( x[ i ] > max ){  // t.y. prielaida neteisinga – turim jai pakeisti reikšmę
      max = x[ i] ;
      k = i + 1;         // ... indeksui
      }
   }
}
```

Jei paryškintoje eilutėje vietoj ženklo „>“ įrašytume ženklą „<“ – algoritmas ieškotų mažiausiojo elemento masyve ir jo matematinio indekso.

4 pavyzdys. Masyvo rūšiavimas didėjančia tvarka. Čia užprogramuosim tokį algoritmą: surasim didžiausią masyvo elementą ir jį sukeisim su paskutiniu masyvo elementu. Dabar tvarkysim likusią masyvo dalį be paskutinio elemento ir atliksim lygiai tokius pat veiksmus: jau du didžiausi masyvo elementai atsidurs reikiamose vietose. Dabar imsim tvarkyti masyvo dalį be dviejų paskutinių elementų, ir t.t. Algoritmą baigsim, kai masyve liks sutvarkyti tik du pirmuosius masyvo elementus: t.y. *n* elementų turinčiame masyve didžiausio elemento paiešką turėsim atlikti *n-1* kartų.

```
void ascending_order1( double x[ ], int n ){
   //
   //Argumentai:      x - pradinis masyvas; darbo metu pakeičiamas: rūšiuojamas didėjimo
    //                  tvarka
   //                 n – masyvo elementų skaičius
   //
   for( int ii = 0; ii < n-1; ii++ ) {    // n-1 karta kartojama didžiausio elemento paieška vis
       double max = x[ 0 ];          //  trumpesniame masyve...
       int imax = 0;
       for( int i = 1; i < n-ii; i++ ) { //...pirmąkart elementų bus n, toliau n-1, ..., 2
          if( max < x[ i ] ) {
              max = x[ i ];
              imax = i;
          }
       }
       double t = x[ n-ii-1 ];     // sukeičiamas paskutinis masyvo elementas (n-1 : C indeksas!)
       x[ n-ii-1 ] = x[ imax ];    // su didžiausiu elementu per tarpinę ląstelę t
       x[ imax ] = t;
   }
//
}
```

Jei paryškintoje eilutėje vietoj ženklo „<“ įrašytume ženklą „>“ – algoritmas išrikiuotų pradinį masyvą mažėjančia tvarka.

5 pavyzdys. Rūšiavimas kitu algoritmu: imama dviejų pirmų dviejų masyvo elementų pora, elementai sulyginami ir, jei reikia, sukeičiami vietomis. Pereinama prie antrojo-trečiojo elemento poros ir kartojami tie pat veiksmai, ir t.t. iki paskutinių dviejų masyvo elementų poros. Po šios veiksmų sekos į masyvo galą bus išplukdytas didžiausias masyvo elementas, o likusi masyvo dalis dar liks neišrūšiuota. Imam masyvo dalį be paskutiniojo elemento, ir atliekam tą pačią veiksmų seką. Tą kartosim, kol liks tik du pirmieji masyvo elementai.

```
void ascending_order2( double x[ ], int n ){
   //
   //Argumentai:      x - pradinis masyvas; darbo metu pakeičiamas: rūšiuojamas didėjimo
    //                  tvarka
   //                 n – masyvo elementų skaičius
   //
   for( int ii = 0; ii < n-1; ii++ ) {
       for( int i = 0; i < n-ii-1; i++ ) { // n-ii-1 : kad indeksu i būtų kreipiamasi iki
                                           //          priešpaskutiniojo elemento
          if( x[ i ] > x[ i+1 ] ) {
              double t = x[ i ];
              x[ i ] = x[ i+1 ];
              x[ i+1 ] = t;

          }
       }
   }
//
}
```

Jei paryškintoje eilutėje vietoj ženklo „>“ įrašytume ženklą „<“ – algoritmas išrikiuotų pradinį masyvą mažėjančia tvarka.

6 pavyzdys. Galima parašyti efektyvesnes 4 ir 5 programų versijas: pradiniai masyvai gali būti tokie, kad jų elementai išsidėstys reikiama tvarka dar nebaigus išorinių ciklų. Tą galima aptikti – tokiu atveju vidiniame cikle nepakliūtume į sąlygos operatoriaus vidų – ir galėtume nutraukti programos vykdymą. Taip perrašysime 5-ąją programą.

```
void ascending_order3( double x[ ], int n ){
   //
   //Argumentai:      x - pradinis masyvas; darbo metu pakeičiamas: rūšiuojamas didėjimo
   //                  tvarka
   //                 n – masyvo elementų skaičius
   //
   bool b;  // vėliavėlė, kuriai priskirsime true, jei masyvas dar neišrikiuotas
   for( int ii = 0; ii < n-1; ii++ ) {
        b = false;
      for( int i = 0; i < n-ii-1; i++ ) {  // n-ii-1 : kad indeksu i būtų kreipiamasi iki
                                           //          priešpaskutiniojo elemento
         if( x[ i ] > x[ i+1 ] ) {
            b = true;
            double t = x[ i ];
            x[ i ] = x[ i+1 ];
            x[ i+1 ] = t;
         }
      }
      if( !b ) break;   // esant b = false baigti išorinį ciklą, o tuo pačiu – ir programą
   }
//
}
```

Dar vieną rūšiavimo algoritmą („burbulo“) pateiksim rodyklių skyriuje.