14 PASKAITA

Turinys: Bendrieji algoritmai vienmačiams masyvams.
Eilutės duomenys.

Algoritmai

Kadangi I/O išnagrinėta ankstesniuose programų pavyzdžiuose, algoritmai bus apiforminti kaip funkcijos, be jokios įvesties ir išvesties. Pagrindinė programa taip pat neteikiama; joje reikėtų skelbti funkcijos prototipą, įvesti duomenis – masyvo matą ir masyvą, kviesti funkciją, bei išvesti rezultatus – dažniausiai pakeistą masyvą.

1. Duotas vienmatis *double* skaičių masyvas. Pašalinti jo *k*-ąjį (matematinis indeksas) elementą.

```
void del_el( double x[ ], int n, int k ){
        //
        //Argumentai:
        //        x – pradinis masyvas; darbo metu keiciamas
        //        n – jo elementu skaicius
        //        k- salinamas elementas – matematinis elemento indeksas
        //Funkcijai reikia failu <iostream> ir <cstdlib>
        //
        if( k < 1 || k > n ){
                cout<<"Netinkama k reiksme\n";
                exit( 1 );
        }
//
        for( int i = k-1; i < n-1; i++ )
                x[ i ] = x[ i+1 ];
}
```

2. Vienmačiame *double* masyve po *k*-ojo elemento (matematinis indeksas) įterpti naują elementą reikšmės *p*.

```
void ins_el( double x[ ], int n, int k, double p ){
        //
        //Argumentai:
        //        x – pradinis masyvas; darbo metu keiciamas. Pagrindineje programoje
        //            jam skirti n+1 lastele atminties
        //        n – jo elementu skaicius
        //        k-  matematinis elemento, po kurio iterpiama, indeksas
        //        p – iterpiama reiksme
        //Funkcijai reikia failu <iostream> ir <cstdlib>
        //
        if( k < 0 || k > n ){
                cout<<"Netinkama k reiksme\n";
                exit( 1 );
        }
```

```
//
        double t1, t2;  // tarpines lasteles
        t1 = x[ k ];
        x[ k ] = p;
        for( int i = k+1; i < n+1; i++ ){
                t2 = x[ i ];
                x[ i ] = t1;
                t1 = t2;
        }
}
```

3. Vienmačio *double* masyvo rūšiavimas didėjančia tvarka algoritmu: rasti masyve didžiausią elementą, jį nukelti į masyvo galą; apdoroti vienetu trumpesnį masyvą kartojant minėtus veiksmus, …, kol galiausiai bus apdorotas masyvas iš dviejų elementų.

```
void asc_order_1( double x[ ], int n ){
        //
        //Argumentai:
        //      x – pradinis masyvas; darbo metu keiciamas
        //      n – jo elementu skaicius
        //
        for( int ii = 0; ii < n-1; ii++ ){
                double xmax = x[ 0 ]; // max elemento prielaida
                int imax = 0;  // to elemento indekso reiksme
                for( int i = 1; i < n-ii; i++ ){
                        if( xmax < x[ i ]){ // t.y. prielaida neteisinga
                                xmax = x[ i ];
                                imax = i;
                        }
                }
                double t = x[ n-ii –1 ]; //sukeisti galini elementa su max
                x[ n-ii-1 ]= x[ imax ];
                x[ imax ] = t;
        }
}
```

4. Vienmačio double masyvo rūšiavimas didėjančia tvarka algoritmu: lyginti du gretimus elementus ir, jei pirmasis jų didesnis – sukeisti vietomis; pereiti prie kitos elementų poros, ir t.t iki paskutiniosios poros. Tai “nuplukdys” didžiausiąjį elementą į masyvo galą, o likusi masyvo dalis liks neišrūšiuota. Kartoti algoritmą vienetu trumpesniam masyvui, iki liks masyvas iš dviejų elementų.

```
void asc_order_2( double x[ ], int n ){
        //
        //Argumentai:
        //      x – pradinis masyvas; darbo metu keiciamas
        //      n – jo elementu skaicius
        //
        for( int ii = 0; ii < n-1; ii++ ){
                for( int i = 0; i < n-ii-1; i++ ){
                        if( x[ i ] < x[ i+1 ]){
                                double t = x[ i ];
                                x[ i ] = x[ i+1 ];
                                x[ i+1 ] = t
                        }
                }
        }
}
```

Efektyvesnis būtų taip patobulintas algoritmas: jei masyvo elementai išsirūšiuotų reikiama tvarka anksčiau, nei pasibaigtų aprašytasis algoritmas – nutraukti algoritmą:

```
void asc_order_21( double x[ ], int n ){
        //
        //Argumentai:
        //      x – pradinis masyvas; darbo metu keiciamas
        //      n – jo elementu skaicius
        //
        for( int ii = 0; ii < n-1; ii++ ){
                bool b = 0;
                for( int i = 0; i < n-ii-1; i++ ){
                        if( x[ i ] < x[ i+1 ]){
                                b = 1;
                                double t = x[ i ];
                                x[ i ] = x[ i+1 ];
                                x[ i+1 ] = t
                        }
                }
        if( !b ) break;
        }
}
```

Eilutės duomenys

Eilutė saugoma kaip *char* formato vienmatis masyvas arba vidinės C++ klasės *string* objektas.
Čia – apie pirmąjį būdą.
Eilutės struktūra: 1 simbolis – 1B; paskutinysis eilutės simbolis – '\0'. Operacijos << ir >>, taikomos eilutiniams duomenims, veikia iki surandamas šis galinis simbolis arba tarpo simbolis.
Žr. pavyzdį iš 13-os paskaitos medžiagos, failo vardo įvestį. Toks įvedimas – pavojingas, kadangi C++ netikrina, ar neperžengiamos masyvo ribos. Todėl įvedus >15 simbolių be tarpų, įvyks kuri nors atminties klaida. Saugiau įvesti galima su manipuliatoriumi *setw*:

```
. . .
int main( ){
        const int MAX = 16;
        char fv[ MAX ]; // failo vardas – iki 15 simboliu ilgio
        cout<<"Iveskite pradinio failo varda iki 15 simboliu ilgio:\n";
        cin>>setw( MAX )>>fv;
. . .
```

Dabar, kiek simbolių būtų įvesta, vis tiek į masyvą būtų perduoti tik pirmieji 15.

Eilutinės konstantos:
arba *char str[ ] = {'E','i','l','u','t','e' };*
arba tiesiog *char str[ ] = "Eilute";*

Eilutiniai duomenys su tarpų simboliais ir esantys keliose eilutėse. Minėta, kad >> ir << veikia iki tam tikrų simbolių. Norint įvesti bet kokius eilutinius duomenis, teks naudoti *stream* klasės metodą *get*. Metodas perkrautas; galimi tokie argumentų sąrašai:1) tik masyvo vardas, 2) masyvo vardas ir maksimalus masyvo ilgis – apsaugoma nuo masyvo ribų peržengimo, 3) masyvo vardas, ilgis ir simbolis, nurodantis duomens pabaigą – kai duomuo teikiamas keliose eilutėse.
Pavyzdys: eilėraščio iki 1000 simbolių ilgio įvestis galėtų būti tokia:

```
. . .
const int MAX = 1000;
char eil[ MAX ];
cout<<"Iveskite eilerasti";
cin.get( eil, MAX, '$' );
cout<<"Eilerastis:\n"<<eil<<endl;
. . .
```

Eilučių kopijavimas: arba kopijuoti paelemenčiui kiekvieną masyvo elementą iki masyvo pabaigos (masyvo ilgį galima nustatyti funkcija *strlen(eilute)*), arba taikyti funkciją *strcpy(eilute_rezultatas, eilute_pradine)*. Abiems reikia antraštinio failo *cstring*.

Eilučių masyvas – dvimačio masyvo analogas; užrašymo sintaksė truputį skiriasi. Pavyzdys: dvimačio eilučių masyvo formavimas. Į masyvą surašomi savaitės dienų pavadinimai.

```
#include <iostream>
using namespace std;
int main( ){
        const int D = 7,
                  MAX = 15;
                // inicializavimas
        char sd[ D ][ MAX ] = { "Pirmadienis", "Antradienis", "Treciadienis",
                                "Ketvirtadienis", "Penktadienis", "Sestadienis",
                                "Sekmadienis" };
                // kreipinys i atskiras eilutes – spausdinimas
        for( int i = 0; i < D; i++ )
                cout<<sd[ i ]<<endl;
        return 0;
}
```