C++ programavimo kalba

Rodyklė *this*, C++ *string* klasė

(9 paskaita)

Rodyklė *this*

Visos objekto funkcijos gali naudotis rodykle **this**, kuri rodo į patį objektą. Tokiu būdu kiekviena funkcija gali rasti objekto, kuriai ji priklauso adresą.

Pavyzdys:

```
#include <iostream>
using namespace std;
class where
{ private: char c_array[10];
  public:  void address ()
              { cout<< “Mano objekto adresas = “<< this<<endl;  }
};

void main()
{ where w1, w2, w3;
  w1.address ();
  w2.address ();
  w3.address ();
}
```

Rodyklė *this*

Rodyklės *this* pagalba galima pasiekti objekto duomenis.

Pavyzdys:

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

class what
{ private: int alfa;
  public:  void tester ()
              { this->alfa = 11;                    // tas pats kaip   alfa = 11;
                cout<< this->alfa<<endl;  }       // cout<< alfa<<endl;
};

void main()
{ what w1;
  w1.tester ();
}
```

Rodyklės *this* naudojimas su *return*

Praktikoje dažnai rodyklė *this* naudojama, kai reikia grąžinti reikšmes iš funkcijų ar perkrautų operatorių.

Pavyzdys:

```
#include <iostream>
using namespace std;
class alfa
{ private: int data;
  public:  alfa()  { }
           alfa(int d)
           { data = d; }

alfa& operator = (alfa& a)
{ data = a.data;
   cout<< “Iškviestas priskirimo operatorius”<<endl;
   return *this;}

};
```

```
void main()
{
   alfa a1(35);
   alfa a2, a3;
   a3=a2=a1;
 }
```

Rodyklės *this* naudojimo pavyzdys

```
#include <iostream>
using namespace std;
  class Dummy {
   public:
          int is_itme ( Dummy  param);
          {
           if (&param == this)                  // tikrina ar argumentas tas pats objektas
              return 1;
          else
              return 0;
}
void main () {
  Dummy a;
  Dummy* b = &a;
  if ( b->is_itme(a) )                          // tikrina ar argumentas tas pats objektas
    cout << “Taip, &a yra priskirtas rodyklei *b";
}
```

Rodyklės *this* naudojimo pavyzdys

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

class Figura {
  protected:  int ilgis, aukstis;
  public:
        void set_values (int a, int b)
         { ilgis = a; plotis = b; }

        virtual int area (void) =0;

      void print_area (void)
      { cout << this->area() << endl; }
  };
```

```
class Staciakampis: public Figura {
  public:
    int area (void)
      { return (width * height); }
  };

class Trikampis: public Figura {
  public:
    int area (void)
      { return (width * height / 2); }
  };
```

Rodyklės *this* naudojimo pavyzdys

```
int main () {
  Figura * ppoly1 = new Staciakampis;
  Figura * ppoly2 = new Trikampis;

  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);

  ppoly1->printarea();          // anksčiau buvo ppoly1->area();
  ppoly2->printarea();          // anksčiau buvo ppoly2->area();

  delete ppoly1;
  delete ppoly2;

  return 0;
}
```

Rodyklės *this* naudojimo pavyzdys

```
void main () {
 char fi; const int N=5;
  Figura  *ppoly[N];
  for (int i = 0; i<N; i++) {
       cout<< "Ką įvedinėsite: T/S:";
       cin>>fi;
       if ( fi=='t‘ || fi=='T‘ )
                { ppoly[i]= new Trikampis;
                  ppoly[i]->set_values(1,2); }
      else
                {ppoly[i]= new Staciakampis;
                  ppoly[i]->set_values(3,4); }
  }
for ( i = 0; i < N; i++)
   ppoly[i]->printarea();
for ( i = 0; i<N; i++)
   delete ppoly[i];
}
```

C-string ir *string* klasė

C++ darbui su eilutėmis galima naudoti C kalbos funkcijas iš bibliotekos *string.h* arba specialiai darbui su eilutėmis sukurta klase **string**. Taip užtikrinamas C/C++ suderinamumas.

```
#include <string.h>          arba    #include<cstring>          // C-string
                                     using namespace std;
```

Klasę **string** naudoti darbui su eilutėmis yra patogiau, nes:

- nereikia rūpintis dinaminiu atminties valdymu didinant mažinant eilutes;
- leidžiama naudoti perkrautus operatorius;
- didesnis našumas ir saugumas;

```
#include <string>                    // string klasė
using namespace std;
```

string klasės konstruktoriai

string klasė turi keletą konstruktorių, kurie leidžia įvairiai sukurti eilutę.

Konstruktoriai:

```
string ();                                                  // tuščia eilutė
string( const string &s);                                   // kopijos konstruktorius
string( const string &s, size_t start, size_t n= npos);     // kopijos konstruktorius
string( const char *cp);                                    // eilutės priskyrimas
string( const char *cp, size_t start, size_t n= npos);      // kopijuoje n simbolių
string( char c);                                            // eilutė – tai vienas simbolis
string( size_t n=npos, char c);                             // eilutė – tai n simbolių
```

Pavyzdžiai

```
string s1;
string s2 (“Batuotas katinas”);
string s3(‘A’);   string s4(5, ‘B’);
```

Pavyzdys

```
#include <iostream>
#include <string>
using namespace std;
void main()
{ string s1("Vyras");
  string s2 = "Moteris";
  string s3;

  s3= s1;
  cout<< "s3=" << s3 << endl;
  s3 = "SEIMA - tai " + s1 ;
  s3 += " ir " + s2;
  cout << s3 << endl;

 s1.swap(s2);                          // apsikeičia s1 ir s2
 cout<< s1 << " arba " << s2 << endl;
}
```

String objektų įvedimas/išvedimas

Įvedimas/išvedimas atliekamas panašiai, kaip C-string. Operatoriai << ir >> pekraunami dirbant su *string* objektais.

```
#include <iostream>
#include <string>
using namespace std;
void main()
{ string full_name, nickname, address;
  string greeting (“Labas ”);
        cout<< “Ivesk varda ir pavarde “;
  getline(cin, full_name);  cout << greetings<< full_name<<endl;
        cout<< “Ivesk nick’a“;
  cin >> nickname;
  greetings += nickname;
        cout << greetings;
        cout<< “Adresas…. Uzbaigti su $ zenklu”);
  getline(cin, address, ‘$’);
        cout << “Tavo adresas ”<< adress<<endl; }
```

Eilutės dydžio keitimas

Metodai size() – rodo eilutės ilgį (užpildytų simbolių skaičių);
capacity() – rodo eilutei išskirtą atminties dydį;
length() – size() sinonimas;
max_size() – rodo maksimalų leistiną eilutės simbolių skaičių;
reserve() – keičiamas eilutei išskirtos atminties dydis;
resize() – keičia (didina/mažina) eiltutės ilgį;
empty() – grąžina true, jei eilutė tuščia.

```
string s6 = “Mano batai buvo du”;
cout<< s6.size()<<endl;
s6.reserve(200);
cout<<s6.capacity()<< endl;
cout<< s6.max_size()<<endl;

if (s6.empty() )                    // empty() – grąžina true, jei eilutė tuščia
   cout << “Eilutė tuščia \n”;
```

string klasės paieškos metodai

Metodas **find()** randa eilutę, simbolį, ar simbolių masyvą einamojoje eiltutėje ir grąžina jos poziciją (*indekso numerį*). Metodo prototipai:

```
size_type find (const string & SearchString, size_type pos = 0) const;
size_type find (const char * str, size_type position, size_type n) const;
size_type find (const char * str, size_type position = 0) const;
size_type find (char c, size_type position = 0) const;
```

Metodas **rfind()** analogiškas find() tik paieška pradedama nuo eilutės galo.

Kiti paieškos metodai:

find_first_of() - randamas pirmas simbolis einamojoje eilutėje iš patiektų sąraše
find_last_of() - randamas paskutinis simbolis einamojoje eilutėje iš patiektų sąraše
find_first_not_of() - randamas pirmas simbolis einamojoje eilutėje, kuris neatitinka iš patiektų sąraše
find_last_not_of() - randamas paskutinis simbolis einamojoje eilutėje, kuris neatitinka iš patiektų sąraše

string objekto paieška

```
#include <iostream>
#include <string>
using namespace std;
void main()
{ string s1 = “Mano batai buvo du, vienas dingo nerandu”;
   int n;
   if ( s1.find(“batai”) >=0)              //  n= 5, nes numeracija pradedama nuo 0.
          cout<< “batai rastas. Pozicija: “<<n<<endl;

   n = s1.find_first_of(“aov”);          //  n = 1
          cout<< “Pirmasis iš aov rastas pozicijoje: “<<n<<endl;

   n = s1.find_first_not_of(“Maov”);      //  n = 2
          cout<< “Pirmasis ne iš Maov rastas pozicijoje: “<<n<<endl;
}
```

string klasės modifikavimo metodai

Metodas **instert()** leidžia įtepti kitą eilutę, simbolių masyvą ar simbolį į einamą eiltutę. Jo prototipai:

```
string& insert (size_type position, const string & str);
string& insert (size_type position, const string & str, size_type pos2=0, n = npos);
string& insert (size_type position, const char * str);
string& insert (size_type position, size_type n, char c);
```

Metodas **replace()** keičia einamojoje eilutėje simbolius į kitus simbolius arba eilutę. Metodo prototipai:

```
string& replace (size_type position, size_type n, const string & str );
string& replace (size_type pos1, size_type n1, const string & str, size_type pos2,
                 size_type n2);
string& replace (size_type position, size_type n1, const char *s, size_type n2);
string& replace (size_type position, size_type n1, const char *s);
string& replace (size_type position, size_type n1, size_type n2, char c);
```

string objekto modifikavimas

```
#include <iostream>
#include <string>
using namespace std;
void main()
{ string s1("Quick! Send for Count Greystone.");
  string s2("Lord");
  string s3("Don't");
  s1.erase(0, 7);                    // ištrina "Quick"
  s1.replace(9, 5, s2);              // pakeičia "Count" su "Lord"
  s1.replace(0,1,"s");               // pakeičia 'S' į 's'
  s1.insert(0, s3);                  // įterpia "Don't" pradžioje
  s1.erase(s1.size() -1, 1);         // ištrina '.'
  s1.append (3, '!');                // prideda gale "!!!"
  int x = s1.find ('  ');
  while (x < s1.size() )
    { s1.replace(x, 1, "/");
       x = s1.find('  ');  }
   cout << s1<<endl; }
```

string objektų kopijavimas ir trinimas

Metodas **copy()** kopijuoja einamąją eilutę (arba jos dalį) į eilutę *c, nuo nurodytos pozicijos. Grąžinamas nukopijuotų simbolių skaičius. **copy()** prototipas:

```
size_t  copy (char *c, size_type n, size_type position) const;
```

Pvz. s3.copy(str, 3, 5); // į eilutę str bus nukopijuoti 3 simboliai iš s3 pradedant nuo 5 pozicijos

---

Metodas **substr()** grąžina einamosios eilutės dalį. Prototipas:

```
string substr (size_type position, size_type n);
```

Pvz. s4 = s3.substr(0, 3); // grąžinama eilutė iš 3 pirmųjų simbolių

Metodas **erase()** trina simbolius einamojoje eilutėje. Prototipas:

```
string erase (size_type position, size_type n);
```

Pvz. s3.erase(0,3); // ištrina pirmus 3 simbolius

string objektų lyginimo metodai

Klasė string turi visą eilę perkrautų operatorių skirtų objektų eilučių palyginimui. Jų prototipas:

```
bool  operator == ( const string & str1, const string & str2);
bool  operator == ( const string & str1, const char* str2);
bool  operator == ( const char* str1, const string & str2);
```

Leidžiami tokie pekrauti operatoriai : != , < , > , <= , >= .

Palyginimui galima naudoti ir metodą **compare().** Jo prototipai:

```
int  compare ( const string & str );
int  compare ( size_t position, size_t n, const string & str );
```

Pvz. s1.compare(s2);

Gražinamos reikšmės:

0 , jei eilutes lygios

1 ,jei s1 didesnė nei s2

-1 ,jei s2 didesnė nei s1

string objektų lyginimas

```
#include <iostream>
#include <string>
using namespace std;
void main()
{ string aName = “Jurgis”;
  string userName;
  cout << “Įveskite vardą”;
  cin >> userName;
  if (userName == aName)
      cout << “Sveikas Jurgi\n”;
 else if (userName < aName)
      cout << “Tu esi prieš Jurgį\n”;
else
      cout << “Tu esi po Jurgio\n”;
int n = userName.compare(0, 2, aName, 0, 2);
cout  <<“ Dvi pirmosios tavo vardo raidės “;
if (n == 0 )
    cout << “sutampa”;
else if (n < 0)
    cout << “eina pries  “;
else
  cout<< “eina po”;
cout << aName.substr(0,2);
cout<<endl;

}
```

Eilutės elementai

```
#include <iostream>
#include <string>
using namespace std;
void main()
{char array [80];
  string word;
       cout << “Įvesk žodį: ”;
       cin >> word;
  int wlen = word.length();
        cout<< “Vienas simbolis: “;
  for (int j = 0; j < wlen; j++ )
        cout << word.at(j);
 //    cout << word[ j ];        galimas ir toks variantas
  word.copy(array, wlen, 0);
  array[wlen] = ‘\0’;
       cout << “\nArray:  ” << array << endl;
}
```