11. Prelucrarea şirurilor de caractere

Obiective

- Folosirea clasei `string` din biblioteca standard C++ pentru tratarea şirurilor de caractere ca obiecte
- Înţelegerea modului în care se realizează diverse operaţii asupra obiectelor de tip `string`
- Folosirea iteratorilor `string`
- Realizarea operaţiilor de intrare şi ieşire în memorie cu `string`-uri.

11.1 Introducere

Clasa template `basic_string` oferă operaţii tipice pentru prelucrare şirurilor de caractere cum ar fi copierea, căutarea etc. Definiţia acestui template şi toate celelalte funcţionalităţi sunt definite în `namespace std`. Tot aici se găseşte şi declaraţia

```
typedef basic_string<char> string;
```

care creează alias-ul `string` pentru `basic_string<char>`.

Un obiect de tip `string` poate fi iniţializat prin argumentul constructorului:

```
string s1("Hello");
```

prin care se creează un obiect de tip `string` dintr-un `const char*` care conţine caracterele din şirul `"Hello"`, cu excepţia terminatorului `'\0'`. Spre deosebire de şirurile de caractere tip char*, obiectele de tip `string` nu trebuie să conţină caracterul NULL pe ultima poziţie.

Al doilea constructor este cel cu două argumente:

```
string s1(8, 'x');
```

care creează un obiect de tip `string` care conţine opt caractere `'x'`.

Pe lângă aceştia doi, clasa `string` mai dispune de un constructor implicit şi de un constructor de copiere.

11.2 Asignarea şi concatenarea `string`-urilor

Programul de mai jos demonstrează asignarea şi concatenarea obiectelor de tip `string`.

Exemplu

test_assign.cpp

```
#include <iostream>

using std::cout;
using std::endl;

#include <string>
using std::string;

int main()
{
  string s1("cat"), s2, s3;
  s2 = s1; //asignare folosind operatorul =
  s3.assign(s1);//asignare folosind functia assign()
  cout << "s1: " << s1 << "\ns2: " << s2
```

```
        << "\ns3: " << s3 << "\n\n";
    //folosirea operatorului []
    s2[0] = s3[2] = 'r';
    cout << "Dupa modificarea lui s2 si s3:\n"
         << "s1: " << s1 << "\ns2: " << s2 << "\ns3: ";
    //functia membra at()
    int len = s3.length();
    for(int x = 0; x < len; ++x)
      cout << s3.at(x);
    //concatenarea
    string s4(s1 + "apult"), s5;
    //folosirea operatorului +=
    s3 += "pet"; //crearea cuvantului "carpet"
    s1.append("acomb");//crearea cuvantului "catacomb"
    //adauga caracterele incepand cu pozitia 4 din s1 la s5
    s5.append(s1, 4, s1.size());
    cout << "\n\nDupa concatenare:\n" << "s1: " << s1
         << "\ns2: " << s2 << "\ns3: " << s3
         << "\ns4: " << s4 << "\ns5: " << s5 << endl;

    return 0;
}
```

Rulând acest program, obţinem următorul rezultat:

```
s1: cat
s2: cat
s3: cat

Dupa modificarea lui s2 si s3:
s1: cat
s2: rat
s3: car

Dupa concatenare:
s1: catacomb
s2: rat
s3: carpet
s4: catapult
s5: comb
```

Instrucţiunea

```
s2 = s1
```

este o asignare între cele două obiecte în urma căreia `s2` este o copie a lui `s1`. Operaţia de asignare poate fi realizată şi prin funcţia `assign` cu un singur parametru prin care se obţine o copie independentă a obiectului original:

```
s3.assign(s1);//asignare folosind functia assign()
```

În acest exemplu, `s1` este obiectul original şi `s3` este copia sa.

Prin operatorul `[]` se pot accesa caracterele care formează un `string`, în timp ce funcţia `at` dă un acces controlat la caracterele `string`-ului pentru că verifică încadrarea indicilor între limite. Instrucţiunea

```
s2[0] = s3[2] = 'r';
```

foloseşte operatorul `[]` pentru a asigna `'r'` lui `s3[2]`, formându-se cuvântul `"car"` şi pentru a asigna `'r'` lui `s2[0]`, formându-se cuvântul `"rat"`.

Funcţia append cu un parametru apelată ca în exemplul

```
s1.append("acomb");
```

concatenează şirul `s1` cu `"acomb"`, în timp ce funcţia append cu trei parametri ca în apelul:

```
s1.append("acomb");
```

adaugă şirului `s5` caracterele `s1.size()` caractere din `s1` începând cu al patrulea element. Funcţia `s1.size()` întoarce numărul caracterelor lui `s1`.

11.3 Subşiruri

Clasa `string` dispune de funcţia `substr` pentru a extrage un subşir dintr-un `string`. Programul următor demonstrează folosirea funcţiei `substr`.

Exemplu

test_substr.cpp

```
#include<iostream>
using std::cout;
using std::endl;

#include <string>
using std::string;

int main()
{
  string s("The airplane flew away.");

  //intoarce subsirul "plane" care incepe la
  //pozitia 7 si are 5 elemente
  cout << s.substr(7, 5) << endl;

  return 0;
}
```

Rulând acest program, obţinem următorul rezultat:

```
plane
```

Primul argument al funcţiei substr specifică începutul subşirului, iar al doilea argument arată numărul de caractere care urmează să fie extrase.

11.4 Caracteristicile obiectelor `string`

Clasa `string` are funcţii care dau informaţii, între altele, despre lungimea, lungimea maximă, capacitatea şirului. Lungimea unui `string` este numărul de caractere stocate la un moment dat în obiect. Capacitatea sa este numărul total de elemente care pot fi stocate fără creşterea memoriei alocate şirului. Lungimea maximă este cea mai mare dimensiune posibilă a unui obiect de tip `string`. Programul de mai jos prezintă modul în care se pot folosi funcţiile care dau caracteristicile obiectelor de tip `string`.

Exemplu

test_characteristics.cpp

```
#include <iostream>
```

```
using std::cout;
using std::endl;
using std::cin;
#include <string>
using std::string;

void printStats(const string&);

int main ()
{
  string s;
  cout << "Statisticile inainte de input:\n";
  printStats(s);

  cout << "\n\nIntroduceti un sir de caractere: ";
  cin >> s; //delimitat de spatii
  cout << "Sirul introdus este: " << s;

  cout << "\nStatisticile dupa input:\n";
  printStats(s);
  s.resize(s.length()+10);
  cout << "\n\nStatisticile dupa resize cu (length+10):\n";
  printStats(s);
  cout << endl;

  return 0;
}

void printStats(const string& str)
{
  cout << "capacitatea: " << str.capacity()
       << "\nlungimea maxima: " << str.max_size()
       << "\nlungimea: " << str.size()
       << "\nsir vid? " << (str.empty() ? "da" : "nu");
}
```

Rulând acest program, obţinem următorul rezultat:

```
Statisticile inainte de input:
capacitatea: 0
lungimea maxima: 1073741820
lungimea: 0
sir vid? da

Introduceti un sir de caractere: tomato soup
Sirul introdus este: tomato
Statisticile dupa input:
capacitatea: 6
lungimea maxima: 1073741820
lungimea: 6
sir vid? nu
```

```
Statisticile dupa resize cu (length+10):
capacitatea: 16
lungimea maxima: 1073741820
lungimea: 16
sir vid? nu
```

Programul declară un `string s` care iniţial este vid şi îl trimite ca argument funcţiei `printStats`. De la tastatură este introdus apoi şirul `"tomato soup"`. Cele două cuvinte sunt delimitate de spaţiu, iar în şirul `s` este încărcat doar cuvântul `"tomato"`.

Funcţia `printStats` primeşte ca parametru o referinţă la un `const string` şi calculează pentru acesta capacitatea prin funcţia `capacity`, lungimea maximă prin funcţia `max_size` şi dimensiunea prin funcţia `size`. În final verifică daca şirul este vid prin funcţia `empty`.

Funcţia

```
s.resize(s.length()+10);
```

redimensionează obiectul `s` prin mărirea dimensiunii sale cu `10` caractere.

11.5 Găsirea caracterelor într-un `string`

Programul de mai jos arată modul în care se pot folosi funcţiile de căutare într-un `string`.

Exemplu

test_find.cpp

```
#include <iostream>
using std::cout;
using std::endl;

#include <string>
using std::string;

int main()
{
  //compilatorul concateneaza toate sirurile
  //intr-un singur string literal
  string s("Valorile dintr-un subarbore stang"
           "\nsunt mai mici decat valoarea din"
           "\nnodul parinte si valorile din"
           "\norice subarbore drept sunt mai mari"
           "\ndecat valorile din nodul parinte");
  //gasirea cuvantului "subarbore"
  cout << "Sirul original:\n" << s
       << "\n\n(find) \"subarbore\" a fost gasit la: "
       << s.find("subarbore")
       << "\n(rfind) \"subarbore\" a fost gasit la: "
       << s.rfind("subarbore");
  //gasirea literei 'p'
  cout << "\n(find_first_of) caracter din \"qpxz\" la: "
       << s.find_first_of("qpxz")
       << "\n(find_last_of) caracter din \"qpxz\" la: "
       << s.find_last_of("qpxz");
```

```
  //gasirea caracterului '-'
  cout << "\n(find_first_not_of) primul caracter care\n"
       << "  nu este continut in \"Valorie dntusbgmcvp\": "
       << s.find_first_not_of("Valorie dntusbgmcvp");
  //gasirea caracterului '\n'
  cout << "\n(find_last_not_of) primul caracter de la dreapta
care\n"
       << "  nu este continut in \"Valorie dntusbgmcvp\": "
       << s.find_last_not_of("Valorie dntusbgmcvp") << endl;

  return 0;
}
```

Rulând acest program, obţinem următorul rezultat:

```
Sirul original:
Valorile dintr-un subarbore stang
sunt mai mici decat valoarea din
nodul parinte si valorile din
orice subarbore drept sunt mai mari
decat valorile din nodul parinte

(find) "subarbore" a fost gasit la: 18
(rfind) "subarbore" a fost gasit la: 103
(find_first_of) caracter din "qpxz" la: 73
(find_last_of) caracter din "qpxz" la: 158
(find_first_not_of) primul caracter care
  nu este continut in "Valorie dntusbgmcvp": 14
(find_last_not_of) primul caracter de la dreapta care
  nu este continut in "Valorie dntusbgmcvp": 132
```

În funcţia main, este declarat şi iniţializat obiectul s. Compilatorul concatenează toate cele cinci şiruri de caractere pentru a forma un singur string literal.

Funcţia find apelată astfel:

```
s.find("subarbore")
```

găseşte poziţia de la care începe cuvântul `"subarbore"` în `string`-ul `s`. Dacă acest şir de caractere nu este găsit, funcţia returnează constanta `string::npos`.

Funcţia

```
s.find("subarbore")
```

întoarce poziţia ultimei apariţii a cuvântului `"subarbore"` în `string`-ul `s`. Similar funcţiei `find`, dacă şirul căutat nu se găseşte în obiectul `s`, atunci funcţia returnează constanta `string::npos`.

Funcţia

```
s.find_first_of("qpxz")
```

găseşte prima apariţie în `s` a oricăruia dintre caracterele care formează şirul `"qpxz"`. Căutarea începe cu prima poziţie din şirul `s`.

Funcţia

```
s.find_last_of("qpxz")
```

găseşte ultima apariţie în `s` a oricăruia dintre caracterele care formează şirul `"qpxz"`, căutarea începând cu ultima poziţie din şirul `s`.

Funcţia

```
s.find_first_not_of("Valorie dntusbgmcvp")
```

găseşte poziţia în `s` a primului caracter care nu face parte din şirul `"Valorie dntusbgmcvp"`. Căutarea începe cu prima poziţie din şirul `s`.

Funcţia

```
s.find_last_not_of("Valorie dntusbgmcvp")
```

găseşte poziţia în `s` a ultimului caracter care nu face parte din şirul `"Valorie dntusbgmcvp"`, iar căutarea începe cu ultima poziţie din şirul `s`.

11.6 Înlocuirea caracterelor dintr-un `string`

Programlu de mai jos demonstrează folosirea funcţiilor de înlocuire şi ştergere a caracterelor dintr-un `string`.

Exemplu

test_replace.cpp

```
#include <iostream>
using std::cout;
using std::endl;

#include <string>
using std::string;

int main()
{
  //compilatorul concateneaza toate sirurile
  //intr-un singur string literal
  string s("Valorile dintr-un subarbore stang"
           "\nsunt mai mici decat valoarea din"
           "\nnodul parinte si valorile din"
           "\norice subarbore drept sunt mai mari"
           "\ndecat valorile din nodul parinte");
  //sterge toate caracterele de la locatia 62
  //pana la sfarsitul sirului
  s.erase(62);
  //afiseaza noul string
  cout << "Sirul original dupa stergere:\n" << s
       << "\n\nDupa inlocuire:\n";
  //inlocuieste toate spatiile cu punct
  unsigned int x = s.find(" ");
  while(x < string::npos)
  {
    s.replace(x, 1, ".");
    x = s.find(" ", x+1);
  }
  cout << s << endl;

  return 0;
}
```

Rulând acest program, obţinem următorul rezultat:

```
Sirul original dupa stergere:
Valorile dintr-un subarbore stang
sunt mai mici decat valoarea
```

```
Dupa inlocuire:
Valorile.dintr-un.subarbore.stang
sunt.mai.mici.decat.valoarea
```

Funcţia

```
s.erase(62);
```

şterge toate elementele de la poziţia 62 până la sfârşitul şirului.

Înlocuirea caracterelor se face prin apelul funcţiei `replace`:

```
while(x < string::npos)
{
  s.replace(x, 1, ".");
  x = s.find(" ", x+1);
}
```

Primul argument al funcţiei `replace` este poziţia de la care se face înlocuirea, al doilea reprezintă numărul de caractere care se înlocuiesc şi al treilea parametru reprezintă şirul de caractere cu care se face înlocuirea. Constanta `string::npos` reprezintă lungimea maximă a `string`-ului. Funcţia `find` returnează această valoare atunci când ajunge la sfârşitul şirului de caractere.

11.7 Inserarea caracterelor într-un `string`

Exemplu

test_insert.cpp

```
#include <iostream>
using std::cout;
using std::endl;

#include <string>
using std::string;

int main()
{
  string s1("inceput sfarsit"),
         s2("mijloc "), s3("12345678"), s4("xx");
  cout << "Sirurile initiale:\ns1: " << s1
       << "\ns2: " << s2 << "\ns3: " << s3
       << "\ns4: " << s4 << "\n\n";
  //insereaza cuvantul "mijloc" la pozitia 8
  s1.insert(8, s2);
  //insereaza "xx" la pozitia 3 din s3
  s3.insert(3, s4, 0, string::npos);
  cout << "Sirurile dupa insert:\ns1: " << s1
       << "\ns2: " << s2 << "\ns3: " << s3
       << "\ns4: " << s4 << endl;

  return 0;
}
```

Rulând acest program, obţinem următorul rezultat:

```
Sirurile initiale:
s1: inceput sfarsit
s2: mijloc
```

```
s3: 12345678
s4: xx

Sirurile dupa insert:
s1: inceput mijloc sfarsit
s2: mijloc
s3: 123xx45678
s4: xx
```

Instrucţiunea

```
s1.insert(8, s2);
```

se foloseşte pentru a insera `string s2` înaintea elementului `8`.

Linia

```
s3.insert(3, s4, 0, string::npos);
```

inserează `s4` înainte de al treilea element al lui `s3`. Ultimele două argumente specifică elementul de start şi numărul caracterelor din `s4` care se inserează.

11.8 Iteratori

Clasa `string` conţine iteratori pentru parcurgerea înainte şi înapoi a unui şir. Iteratorii dau acces individual la caractere, iar sintaxa este similară operatorilor pointer. Programul de mai jos ilustrează folosirea iteratorilor pentru obiecte de tip `string`.

Exemplu

test_iterator.cpp

```
#include <iostream>
using std::cout;
using std::endl;

#include <string>
using std::string;

int main()
{
  string s("Testarea iteratorilor");
  string::const_iterator i1 = s.begin();
  cout << "s = " << s
       << "\n(Folosind iteratorul i1) s este: ";
  while(i1 != s.end())
  {
    cout << *i1;
    ++i1;
  }
  cout << endl;

  return 0;
}
```

Rulând acest program, obţinem următorul rezultat:

```
s = Testarea iteratorilor
(Folosind iteratorul i1) s este: Testarea iteratorilor
```

Funcţia
```
s.begin()
```
returnează un iterator pentru obiectul `s`.

Instrucţiunile
```
while(i1 != s.end())
{
  cout << *i1;
  ++i1;
}
```
folosesc iteratorul `i1` pentru a parcurge conţinutul obiectului `s`. Funcţia `s.end()` returnează un iterator la prima poziţie după ultimul element al lui `s`. Conţinutul fiecărei locaţii este tipărit prin dereferenţierea iteratorului, după care acesta avansează cu o poziţie prin apelul operatorului `++`.

11.9 Procesarea stream-urilor `string`

Suplimentar prelucrării stream-urilor standard I/O şi a stream-urilor de fişiere I/O, limbajul C++ include posibilitatea intrărilor din `string`-uri din memorie şi a ieşiriilor în `string`-uri în memorie. Aceste mecanisme se numesc *in-memory I/O* sau *string stream processing*.

Intrările din `string`-uri sunt suportate de clasa `istringstream`, iar ieşirile în `string`-uri sunt suportate prin clasa `ostringstream`. Aceste două clase sunt, de fapt, două alias-uri definite astfel:
```
typedef basic_istringstream<char> istringstream;
typedef basic_ostringstream<char> ostringstream;
```
Clasele `basic_istringstream` şi `basic_ostringstream` au aceleaşi funcţionalităţi ca `istream` şi `ostream` la care se adaugă câteva funcţii specifice formatării în memorie. Programele care folosesc formatarea în memorie trebuie să includă fişierele header `<sstream>` şi `<iostream>`.

O aplicaţie a acestei tehnici este validarea datelor. Un program poate citi într-o primă fază o linie de date dintr-un stream într-un `string`, urmând ca apoi o rutină de validare să parcurgă şi, dacă este necesar, să corecteze datele. În final, programul poate citi datele din `string` ştiind că acestea sunt în formatul corect.

Ieşirea într-un `string` este o modalitate prin care se poate profita de posibilităţile puternice de formatare de care dispune limbajul C++. Datele pot fi pregătite într-un `string` pentru a simula formatul de afişare pe ecran, după care obiectul de tip `string` poate fi transferat pe disc păstrând imaginea de pe ecran.

Un obiect `ostringstream` foloseşte un obiect `string` pentru a păstra datele care urmează a fi afişate. Funcţia membră `str` a clasei `ostringstream` returnează o copie de tip `string` a conţinutului obiectului.

Programul de mai jos demonstrează modul în care poate fi folosit obiectul `ostringstream`.

Exemplu

test_ostringstream.cpp
```
#include <iostream>
using std::cout;
using std::endl;

#include <string>
using std::string;
```

```
#include <sstream>
using std::ostringstream;

int main()
{
  ostringstream outputString;
  string s1("Iesirie catorva tipuri de date "),
         s2("intr-un obiect ostringstream:"),
         s3("\n       double: "),
         s4("\n          int: "),
         s5("\nadresa de int: ");
  double d = 123.4567;
  int i = 22;

  outputString << s1 << s2 << s3 << d << s4 << d << s5 << &i;
  cout << "outputString contine:\n" << outputString.str();

  outputString << "\nse mai adauga caractere";
  cout << "\n\nDupa inserarea in stream,\n"
       << "outputString contine:\n" << outputString.str()
       << endl;

  return 0;
}
```

Rulând acest program, obţinem următorul rezultat:

```
outputString contine:
Iesirie catorva tipuri de date intr-un obiect ostringstream:
       double: 123.457
          int: 123.457
adresa de int: 0x22fe34

Dupa inserarea in stream,
outputString contine:
Iesirie catorva tipuri de date intr-un obiect ostringstream:
       double: 123.457
          int: 123.457
adresa de int: 0x22fe34
se mai adauga caractere
```

Un obiect `istringstream` citeşte date dintr-un `string` în memorie. Datele sunt păstrate într-un `istringstream` ca şi caractere. Intrările din obiectele `istringstream` funcţionează asemănător intrării dintr-un fişier în general, sau de la intarea standard în particular. Sfârşitul obiectului `string` este interpretat de `istringstream` ca *end-of-file*.

Programul următor ilustrează modalitatea în care se pot folosi obiecte `istringstream`.

Exemplu

test_istringstream.cpp

```
#include <iostream>
using std::cout;
```

```
using std::endl;

#include <string>
using std::string;
#include <sstream>
using std::istringstream;

int main()
{
  string input("Input test 123 4.7 A");
  istringstream inputString(input);
  string string1, string2;
  int i;
  double d;
  char c;

  inputString >> string1 >> string2 >> i >> d >> c;

  cout << "Au fost extrase urmatoarele elemente\n"
       << "din obiectul istringstream:"
       << "\nstring: " << string1
       << "\nstring: " << string2
       << "\n   int: " << i
       << "\ndouble: " << d
       << "\n  char: " << c;
  //citire din stream vid
  long x;
  inputString >> x;
  if(inputString.good())
    cout << "\n\nvaloarea long este: " << x << endl;
  else
    cout << "\n\ninputString este vid" << endl;

  return 0;
}
```

Rulând acest program, obţinem următorul rezultat:

```
Au fost extrase urmatoarele elemente
din obiectul istringstream:
string: Input
string: test
   int: 123
double: 4.7
  char: A

inputString este vid
```

Obiectul `input` de tip `string` este iniţializat cu valoarea

```
Input test 123 4.7 A
```

ale cărui elemente sunt citite în variabilele `string1`, `string2`, `i`, `d` şi `c` prin instrucţiunea

```
inputString >> string1 >> string2 >> i >> d >> c;
```

Aceste valori sunt apoi afişatepe ecran. În final, programul încearcă să citească din inputString care este vid pentru că tot conţinutul său a fost deja trimis către cout prin instrucţiunea anterioară. Condiţia

```
inputString.good()
```

este evaluată ca fiind `false`, executându-se ramura `else` a structurii `if/else`.