Clasa *string* din biblioteca standard C++

1. Reprezentarea seturilor de caractere

Utilizatorii unui program se aşteaptă ca acesta să se supună convenţiilor culturale naţionale (precum modul de afişare al datei, valorile monetare, modul de reprezentare al numerelor, sistemul de măsură, etc.). De asemenea, seturile de caractere au proprietăţi distincte şi pot avea codări diferite, ceea ce necesită soluţii flexibile pentru a rezolva probleme precum "ce este considerat a fi o literă?" sau "ce tip se utilizează pentru a reprezenta caractere?".

Pentru a gestiona seturile de caractere care posedă mai mult de 256 de elemente, se utilizează două tipuri de reprezentări: *multibyte* şi *wide-character*. În reprezentarea multibyte, numărul de octeţi utilizaţi pentru un caracter este variabil; un caracter ISO Latin-1 (reprezentat pe un octet) poate fi urmat, de exemplu, de o ideogramă japoneză (reprezentată pe 3 octeţi). Astfel, la un moment dat, un octet poate fi privit fie ca reprezentând un caracter, fie doar o parte a unui caracter. Pentru a rezolva această problemă, toţi octeţii se interpretează conform setului de caractere curent (starea curentă), până la întâlnirea unui caracter *escape*; următorul caracter are rolul are rolul de a semnaliza care este noul set de caractere (noua stare) după care se vor interpreta octeţii în continuare. În reprezentarea wide-character, numărul de octeţi utilizaţi pentru un caracter este acelaşi (de obicei, 2 sau 4); această reprezentare nu diferă, conceptual, de reprezentările pe un octet. Reprezentarea multibyte este mai compactă şi este utilizată, în mod obişnuit, pentru scrierea datelor în afara programelor; este mult mai uşor de procesat caractere de dimensiune fixă, ceea ce face ca reprezentarea wide-character să fie utilizată, în mod obişnuit, în interiorul programelor. C++ si C utilizează în acest scop tipul *wchar_t*; spre deosebire de C, în C++ wchar_t este un tip predefinit şi nu un alias; este astfel posibilă supraîncărcarea funcţiilor pe baza acestui tip.

Diferitele reprezentări ale seturilor de caractere implică variaţii care sunt relevante în procesarea şirurilor de caractere (detaliile în care diferă operaţii uzuale precum atribuirea, copierea sau compararea) şi în operaţiile I/O (modul în care se reprezintă EOF). Este de dorit instanţierea claselor de tip string şi stream cu tipuri predefinite, în special char şi wchar_t; însă interfaţa tipurilor predefinite nu poate fi modificată. De aceea, aspectele ce ţin de reprezentare sunt gestionate de o *clasă caracteristică* (en. trait); clasele string şi stream sunt şabloane care iau ca argument o clasă caracteristică. De exemplu:

```
namespace std{
template<class charT> struct char_traits {
        ...
};

template<class charT, class traits = char_traits<charT>, class Allocator = allocator<charT> > class basic_string;

template<class charT, class traits = char_traits<charT> > class basic_istream;
template<class charT, class traits = char_traits<charT> > class basic_ostream;
}
```

O clasă caracteristică defineşte proprietăţile fundamentale ale unui tip caracter şi operaţiile fundamentale pentru implementarea string-urilor şi a stream-urilor pe baza acestui tip:

Expresie	**Semnificaţie**
char_type	Tipul caracter.
int_type	Un tip întreg, suficient de larg pentru a cuprinde şi valoarea suplimentară EOF.
pos_type	Un tip utilizat pentru a reprezenta poziţii în stream-uri.
off_type	Un tip utilizat pentru a reprezenta distanţele (en. offset) între poziţii din stream-uri.

state_type	Un tip utilizat pentru a reprezenta starea curentă într-un stream multibyte.
assign(c1, c2)	Atribuie caracterul c2 lui c1.
eq(c1, c2)	Testează dacă caracterele c1 şi c2 sunt egale.
lt(c1, c2)	Testează dacă caracterul c1 este mai mic decât caracterul c2.
length(s)	Returnează lungimea şirului s.
compare(s1, s2, n)	Compară şirurile s1 şi s2, considerând până la n caractere.
copy(s1, s2, n)	Copie n caractere din şirul s2 în şirul s1.
move(s1, s2, n)	Copie n caractere din şirul s2 în şirul s1; s1 şi s2 se pot suprapune.
assign(s, n, c)	Atribuie de n ori caracterul c sirului s.
find(s, n, c)	Returnează un pointer la primul caracter din sirul s care este egal cu c, sau 0 dacă nu există un astfel de caracter în primele n din şir.
eof()	Returnează valoarea EOF.
to_int_type(c)	Converteşte caracterul c în reprezentarea sa int_type.
to_char_type(i)	Converteşte reprezentarea int_type a lui i într-un caracter; rezultatul convertirii lui EOF este nedefinit.
not_eof(i)	Returnează valoarea i dacă aceasta nu este EOF; altfel, returnează o valoare dependentă de implementare.
eq_int_type(i1, i2)	Testează egalitatea a două caractere i1 şi i2 în reprezentarea int_type (deci i1 şi i2 pot fi EOF).

În unele reprezentări, tipul caracter nu include şi valoarea EOF. Limbajul C a impus convenţia ca funcţiile care citesc caractere să returneaza int (şi nu char); C++ a extins această tehnică. *char_type* este tipul care reprezintă toate caracterele, în timp ce *int_type* este tipul care reprezintă toate caracterele, plus EOF; evident, cele două tipuri pot fi identice.

Biblioteca standard include specializări ale şablonului *char_traits*<> pentru tipurile char şi wchar_t; aceste specializări sunt implementate, în mod uzual, utilizând funcţiile C din <cstring>:

```
namespace std{
        template<> struct char_traits<char>;
        template<> struct char_traits<wchar_t>;
}
```

2. Descrierea clasei *string*

Toate funcţiile şi definiţiile de tip pentru şirurile de caractere sunt introduse de headerul standard <string> (a nu se confunda cu fişierul header din biblioteca C standard, <cstring>); tipul şablon *basic_string*<> este tipul de bază pentru toate clasele care gestionează şiruri de caractere:

```
namespace std{
template<class charT, class traits = char_traits<charT>, class Allocator = allocator<charT> > class basic_string;
}
```

Primul argument al şablonului reprezintă tipul de dată al unui caracter. Al doilea argument este clasa caracteristică care defineşte proprietăţile tipului caracter utilizat. Al treilea argument defineşte modelul de memorie utilizat; pentru aceasta se foloseşte un tip special, numit *alocator*. Un obiect alocator traduce o cerere de alocare de memorie într-un apel intern; astfel, fără a schimba interfaţa, se pot utiliza politici diferite de gestionare a memoriei: shared memory, garbage collection, etc. Biblioteca standard defineşte tipul *allocator* ca un şablon care poate fi utilizat ca argument implicit.

Două specializări ale clasei basic_string<> sunt oferite în biblioteca standard:

```
namespace std{
        typedef basic_string<char> string;
        typedef basic_string<wchar_t> wstring;
}
```

Se pot utiliza deci obiecte şir de caractere care manipulează orice fel de seturi de caractere; interfața este aceeaşi! În cele ce urmează, vom discuta doar despre *string-şirurile de caractere*, numite în continuare obiecte string sau string-uri.

Clasa *string* oferă foarte multe metode de manipulare a şirurilor de caractere; unele dintre acestea sunt implementate prin funcții supraîncărcate, care necesită unul, două sau trei argumente. Toate aceste funcții respectă următoarea schemă de interpretare a argumentelor:

Argumente	**Semnificație**
const string& s	Întregul string s.
const string& s, size_type poz, size_type n	Cel mult primele n caractere din string-ul s, începând de la poziția poz.
const char* cs	Întregul C-şir cs.
const char* cs, size_type n	n caractere din tabloul cs.
char c	Caracterul c.
size_type n, char c	De n ori caracterul c.
iterator beg, iterator end	Toate caracterele din domeniul [beg, end).

Doar versiunea cu un singur argument char* îl tratează pe '\0' ca un caracter special, care termină un C-şir; în toate celelalte cazuri, '\0' nu are o semnificație specială. Astfel, un string poate conține orice caracter.

Clasa *string* dă posibilitatea tratării şirurilor de caractere de o manieră sigură şi intuitivă, deoarece a fost proiectată să se comporte precum un tip fundamental: se pot face atribuiri cu =, comparări cu ==, concatenări cu + şi se pot căuta şi extrage subşiruri, toate acestea fără a mai avea grija alocării şi gestionării memoriei.

3. Constructori şi destructori. Accesul la elemente şi dimensiunea şirurilor

string::string()	String vid.
string::string(const string& s)	
string::string(const string& s, size_type poz)	
string::string(const string& s, size_type poz, size_type n)	Inițializează obiectul string cu cel mult primele n caractere din string-ul s, începând de la poziția poz; dacă n lipseşte, se utilizează toate caracterele din s, începând de la poziția poz.
string::string(const char* cs)	Inițializează cu caracterele C-şirului cs, mai puțin '\0'.
string::string(const char* cs, size_type n)	Inițializează cu n caractere din tabloul cs; '\0' nu are semnificație specială; în cs trebuie să existe cel puțin n caractere.
string::string(size_type n, char c)	Inițializează cu de n ori caracterul c.
string::string(InputIterator beg, InputIterator end)	Pentru compatibilitate cu STL.
string::~string()	Eliberează memoria.

char& string::operator[] (size_tipe i) char string::operator[] (size_tipe i) const	Ambele forme returnează caracterul de la poziția i. Pasarea unui index invalid conduce la comportament nedefinit.
char& string::at(size_type i) const char& string::at(size_type i) const	Pentru ambele forme pasarea unui index invalid generează excepția out_of_range.

size_type string::length() const size_type string::size() const	Returnează numărul de caractere. Versiune echivalentă, oferită pentru compatibilitate cu STL.
bool string::empty() const	Mai rapidă decât string::size()==0!
size_type string::max_size() const	Returnează numărul maxim de caractere pe care le poate conține un string.
size_type string::capacity() const	Returnează numărul de caractere pe care un string le poate păstra fără realocare (adică dimensiunea curentă a buffer-ului intern).
void string::reserve()	Micşorează capacitatea buffer-ului intern la valoarea returnată de length().
void string::reserve(size_type n)	Rezervă memorie pt. cel puțin n caractere. Dacă n<length(), analog reserve().

Se observă că string-urile pot fi inițializate în diverse forme care oferă flexibilitate şi control. Astfel, se pot utiliza pentru inițializare alte obiecte string, C-şiruri, portiuni din C-şiruri sau din obiecte string, tablouri de caractere, literali. Nu se pot utiliza pentru inițializare valori intregi sau char; aceasta deoarece nu există constructor care să aibă un argument de tip char (ci numai de tip const char*).

```
string blank1;                                  //şir vid
string blank2 = "";                             //şir vid
string eroare1('a');                            //eroare
string eroare2(1);                              //eroare
string tablou_un_caracter(1,'a');               //a
string tablou_10_caractere(10, 'a');            //aaaaaaaaaa
string carte("Lord of the rings");
string autor = "J.R.R. Tolkien";
string copie_carte = carte;
string alta_copie_carte(carte.begin(), carte.end());
string nume(autor, 7, 7);                       //Tolkien
string prima_initiala(autor, 0,1);              //J
```

Tipul întreg fără semn *size_type* este utilizat pentru valori ce reprezintă indexuri şi dimensiuni; constanta statică *string::npos* de tip size_type, inițializată cu -1 (valoarea maximă), are, în funcție de context, semnificațiile "index invalid" sau "restul caracterelor din şir". Datorită primei semnificații, dimensiunea maximă a unui obiect string este npos-1; dacă, în urma unor diverse operații, rezultă un şir de lungime mai mare decât max_size(), este generată excepția *length_error*.

Dimensiunea curentă a buffer-ului intern al unui string este returnată de capacity(); cunoaşterea dimensiunii este importantă deoarece o realocare a buffer-ului intern consumă timp şi, mai grav, invalidează referințele, pointerii şi iteratorii care se referă la caracterele din string. Dacă se ştie de la început capacitatea maximă, se poate utiliza reserve() pentru a se aloca buffer-ului intern numărul dorit de octeți:

```
blank1.reserve(100);
cout << carte.length() << '\t' << carte.capacity() << '\t' << carte.max_size() << endl;
cout << blank1.length() << '\t' << blank1.capacity() << '\t' << blank1.max_size() << endl;
```

Operatorul [i] şi funcția membru at(i) returnează caracterul de pe poziția i; spre deosebire de [], at() verifică dacă indexul i este valid şi generează excepția *out_of_range* în caz contrar; apelarea operatorului [] cu un index invalid conduce la un comportament nedefinit. Spre deosebire de C-şiruri, obiectele string nu se bazează pe '\0' ca terminator de şir; ele pătrează lungimea, pe care o

raportează length() şi size(). Din acest motiv, ultimul caracter din şir se află pe poziția length()-1; ca de obicei, primul caracter se află pe poziția 0. Însă, pentru versiunea const a operatorului [], length() este un index valid iar operatorul returnează caracterul '\0' (adică valoarea generată de constructorul implicit char()).

```
const string vrajitor1("Gandalf cel Sur");
string vrajitor2("Saruman cel alb");
char c=vrajitor2[0];                    //c conține 'S'
c=vrajitor2.at(1);                      // c conține 'a'
c=vrajitor2.[100];                      //eroare; comportament nedefinit
c=vrajitor2.at(100);                    //genereaza exceptia out_of_range
c=vrajitor2[vrajitor2.length()];        // eroare; comportament nedefinit
c= vrajitor1[vrajitor1.length()];       //c conține '\0'
c=vrajitor2.at(vrajitor2.length());     // genereaza exceptia out_of_range
c=vrajitor1.at(vrajitor1.length());     // genereaza exceptia out_of_range
vrajitor2[12] = 'A';                    //Saruman cel Alb
```

4. String-uri şi C-şiruri

Se pot utiliza C-şiruri şi string-uri aproape în orice fel de combinație; însă, din motive de securitate, nu se oferă conversii automate între *string* şi *char**. Clasa *string* oferă, în schimb, metode de creare şi de scriere/citire a C-şirurilor (cu observația că, pentru un obiect string, '\0' este un caracter obişnuit).

const char* string::c_str() const	Returnează conținutul string-ului sub forma unui C-şir.
const char* string::data() const	Returnează conținutul string-ului sub forma unui tablou de caractere (nu adaugă '\0').
size_type string::copy(char* buf, size_type buf_size) const size_type string::copy(char* buf, size_type buf_size, size_type i) const	Copie, în tabloul buf, cel mult buf_size caractere din string, începând de la poziția i (nu adaugă '\0').

Se observă că c_str() şi data() returnează un C-şir constant. Aceasta însemnă că C-şirul este proprietatrea obiectului string şi este valid doar atâta timp cât string-ul există şi, pentru el, se apelează numai funcții const. De asemenea, C-şirul nu poate fi modificat sau şters (cu free() sau delete).

Copy() returnează numărul caracterelor transferate dintr-un string într-o zonă de memorie oferită de utilizator; acesta trebuie să se asigure că dimensiunea tabloului buf este suficient de mare, în caz contrar comportamentul fiind nedefinit. Se generează excepția out_of_range dacă i>size().

```
string duhuri_inele("9");
int nr = atoi(duhuri_inele.c_str());
string regate("Gondor");
const char* p = regate.c_str();
int dim = strlen(p);                    //p este valid
regate += "si Arnor";
dim = strlen(p);                        //eroare! p este invalid
```

5. Subşiruri şi concatenări

string string::substr() const string string::substr(size_type i) const string string::substr(size_type i, size_type n) const	Returnează o copie a string-ului. Returnează subşirul ce începe la poziția i. Returnează un subşir format din cel mult n caractere, începând de la poziția i.
string operator+(const string& s1, const string& s2) string operator+(const string& s1, const char* s2) string operator+(const char* s1, const string& s2) string operator+(const string& s, char c) string operator+(char c, const string& s)	Toate cele 5 forme returnează un obiect string care conține caracterele din cei 2 operanzi. Dacă sunt mai mult de max_size() caractere, se generează excepția length_error.

Dacă i>size(), substr() generează excepția out_of_range.

6. Modificarea obiectelor string

string& string::operator= (const string& s) string& string::assign (const string& s)	
string& string::assign (const string& s, size_type i, size_type n)	Generează excepția out_of_range dacă i>s.size()
string& string::operator= (const char* s)	Atribuie toate caracterele (mai puțin '\0') din C-şirul s; s trebuie să fie diferit de NULL. Dacă s conține mai mult de max_size()
string& string::assign (const char* s)	caractere, ambele forme generează excepția length_error.
string& string::assign (const char* s, size_type n)	Atribuie n caractere dintr-un tablou; '\0' nu are semnificație specială.
string& string::operator= (char c) string& string::assign (size_type n, char c)	Atribuie n apariții ale caracterului c.
void string::swap(string& s) void swap(string& s1, string& s2)	Ambele forme interschimbă valorile a două obiecte string. Sunt preferabile atribuirii (dacă e posibil), fiind mai rapide.
string& string::operator+=(const string& s) string& string::append(const string& s)	Generează excepția length_error dacă rezultatul are mai mult de max_size() caractere.
string& string::append(const string& s, size_type i, size_type n)	Generează execepția out_of_range dacă i>s.size().
string& string::operator+=(const char* s) string& string::append(const char* s)	s trebuie să fie diferit de NULL.
string& string::append(const char* s, size_type n)	'\0' nu are semnificație specială.
string& string::operator+=(char c) void string::push_back(char c) string& string::append(size_type n, char c)	Compatibilitate cu STL.
string& string::append(InputIterator beg, InputIterator end)	Adaugă toate caracterele din intervalul [beg, end)

Pentru a modifica un string-şir se pot utiliza operatorul de atribuire şi metoda assign(); aceasta din urmă este utilizată atunci când, pentru specificarea valorii atribuite, sunt necesare mai multe argumente. Analog, pentru adăugarea de caractere la sfârşit, se pot utiliza operatorul += şi metoda append(); metoda push_back() este oferită suplimentar, pentru compatibilitate cu STL.

```
string vrajitor("Gandalf cel Sur");
string s;
s = vrajitor;                          // analog, s.assign(vrajitor);
s = "doua\nranduri";                   //analog, s.assign("doua\nranduri");
s.assign(vrajitor, 8, 3);              //atribuie "cel"
s.assign(vrajitor, 8, string::npos);   //atribuie "cel Sur"
s.assign("Gandalf", 8);                //atribuie 'G' 'a' 'n' 'd' 'a' 'l' 'f' '\0'
s.assign(5, 'a');                      //atribuie 'a' 'a' 'a' 'a' 'a'
string alt_vrajitor("Saruman");
string tmp(" cel Alb");
alt_vrajitor += tmp;                   //analog, alt_vrajitor.append(" cel Alb");
alt_vrajitor.push_back('\0');
swap(vrajitor, alt_vrajitor);
```

string& string::insert(size_type i, const string& s)	Inserează caracterele din s în obiectul curent, începând de la poziția i; dacă i>size(), generează excepția out_of_range; dacă rezultatul depăşeşte max_size(), se generează excepția length_error.
string& string::insert(size_type i, const string& s, size_type si, size_type n)	Inserează cel mult n caractere din s, începând de la poziția si, în obiectul curent, începând de la poziția i.
string& string::insert(size_type i, const char* s)	s trebuie să fie diferit de NULL; nu se inserează '\0'.
string& string::insert(size_type i, const char* s, size_type n)	Inserează n caractere din tabloul s (s diferit de NULL); '\0' nu are semnificație specială.
string& string::insert(size_type i, size_type n, char c)	Inserează de n ori caracterul c începând de la poziția i.
void string::insert(iterator poz, size_type n, char c)	Inserează de n ori caracterul c începând de la poziția anterioară celei la care se referă iteratorul poz.
iterator string::insert(iterator poz, char c)	Inserează caracterul c la poziția anterioară celei la care se referă iteratorul poz. Returnează poziția caracterului inserat.
void string::insert(iterator poz, InputIterator beg, InputIterator end)	Inserează toate caracterele din intervalul [beg, end) începând de la poziția anterioară celei la care se referă iteratorul poz.

Se observă că funcția membru insert() nu este supraîncărcată pentru a accepta un index şi un caracter; din acest motiv, un caracter trebuie pasat ca un C-şir. Se poate utiliza însă varianta în care caracterul inserat se repetă de n ori (în acest caz, n==1). Dacă poziția de inserare este 0, datorită conversiilor echivalente ale lui 0 în size_type şi iterator, avem ambiguitate.

```
string vrajitor("ndalf");
vrajitor.insert(0, 'a');                        //eroare
vrajitor.insert(0, "a");                        //"andalf"
vrajitor.insert(0, 1, 'G');                     //eroare; ambiguitate in conversia lui 0
vrajitor.insert((string::size_type)0, 1, 'G'); //conversia elimină ambiguitatea; "Gandalf"
```

string& string::replace(size_type i, size_type n, const string& s)	Înlocuieşte cel mult n caractere din obiectul curent, începând de la poziția i, cu toate caracterele din s.
string& string::replace(iterator beg, iterator end, const string& s)	Înlocuieşte toate caracterele din intervalul [beg, end) din obiectul curent cu toate caracterele din s.
string& string::replace(size_type i, size_type n, const string& s, size_type is, size_type ns)	Înlocuieşte cel mult n caractere din obiectul curent, începând de la poziția i, cu cel mult ns caractere din s, începând de la poziția is.
string& string::replace(size_type i, size_type n, const char* s)	s nu trebuie să fie NULL; '\0' nu este inserat în obiectul curent.
string& string::replace(iterator beg, iterator end, const char* s)	s nu trebuie să fie NULL; '\0' nu este inserat în obiectul curent.
string& string::replace(size_type i, size_type n, const char* s, size_type d)	s nu trebuie să fie NULL; '\0' nu are semnificație specială.
string& string::replace(iterator beg, iterator end, const char* s, size_type d)	s nu trebuie să fie NULL; '\0' '\0' nu are semnificație specială.
string& string::replace(size_type i, size_type n, size_type d, char c)	
string& string::replace(iterator beg, iterator end, size_type d, char c)	
string& string::replace(iterator beg, iterator end, InputIterator newBeg, InputIterator newEnd)	

```
string vrajitor("Gemlf");
vrajitor.replace(1,2, "anda");                    //Gandalf
```

void string::clear()	Şterge toate caracterele din obiectul curent.
string& string::erase()	Analog; returnează referință la obiectul curent, *this.
string& string::erase(size_type i)	Şterge toate caracterele, începând de la poziția i.
string& string::erase(size_type i, size_type n)	Şterge cel mult n caractere, începând de la poziția i.
string& string::erase(iterator poz)	Şterge caracterul de la poziția poz.
string& string::erase(iterator beg, iterator end)	Şterge caracterele din intervalul [beg, end).
void string::resize(size_type n)	Modifică numărul caracterelor din obiectul curent în n. Dacă n<size(), se şterg caractere de la sfârşit. Dacă n>size() se adaugă la sfârşit de (n-size()) ori caracterul '\0'.
void string::resize(size_type n, char c)	Analog; dacă n>size(), se adaugă la sfârşit de (n-size()) ori caracterul c.

8. Căutarea caracterelor şi a subşirurilor

Clasa *string* oferă foarte multe metode care permit căutarea unui caracter, a unui subşir (secvență de caractere) sau a unui caracter dintr-un anumit set. Căutarea se poate desfăşura "înainte" sau "înapoi" şi poate începe de la orice poziție.

Numele tuturor acestor metode începe cu "find"; cum anume se desfăşoară căutarea depinde de numele exact al metodei. Toate metodele de căutare returnează indexul primului caracter găsit; dacă căutarea eşuează, ele returnează *string::npos*.

Nume metodă de căutare	Efect
find()	Caută prima apariție.
rfind()	Caută prima apariție începând de la sfârşit.
find_first_of()	Caută primul caracter dintr-un set dat.
find_last_of()	Caută ultimul caracter dintr-un set dat.
find_first_not_of()	Caută primul caracter care nu este într-un set dat.
find_last_not_of()	Caută ultimul caracter care nu este într-un set dat.

Primul argument este întotdeauna valoarea căutată; al doilea argument, opțional, indică indexul de la care începe căutarea; al treilea argument, opțional, este numărul caracterelor din valoarea căutată. Fiecare dintre aceste metode este supraîncărcată astfel încât să accepte următoarele seturi de argumente:

- const string& ss -> caută subşirul ss
- const string& ss, size_type i -> caută subşirul ss, începând de la poziția i
- const char* cs -> caută C-şirul cs
- const char* cs, size_type i -> caută C-şirul cs, începând de la poziția i
- const char* s, size_type i, size_type n -> caută n caractere din tabloul s, începând de la poziția i ('\0' nu are semnificație specială)
- const char v -> caută caracterul v
- const char v, size_type i-> caută caracterul v, începând de la poziția i

```
string motto = "Trei inele pentru stapanii elfi cei de sub soare,"
        "Sapte pentru ei, piticii de vita din sali de stanca,"
        "Noua, Oamenilor care stiu ca-n lumea lor se moare,"
        "Unul pentru el, Seniorul Intunecimii-n noaptea lui adanca"
        "Unde-s Umbrele in Tinutul Mordor, ca sa le gaseasca."
        "Si pe toate sa le-adune un inel, si altul nime,"
        "Sa le ferece pe toate, astfel sa le stapaneasca,"
        "Unde-s Umbrele, in Tinutul Mordor, in intunecime.";

motto.find("Mordor");               //returneaza 234
motto.find("ei", 4);                //returneaza 33
motto.rfind("Mordor");              //returneaza 382
motto.find_first_of("inel");        // returneaza 2
motto.find_last_of("inel");         // returneaza 402
motto.find_first_not_of("inel");    // returneaza 0
motto.find_last_not_of("inel");     // returneaza 403
motto.find("abc");                  // returneaza string::npos
```

Atunci când căutarea eşuează, valoarea returnată este *string::npos*. Aceasta este o constantă de tip *size_type*, inițializată cu valoarea -1; cum size_type este un tip întreg fără semn, string::npos reprezintă valoarea maximă reprezentabilă. Din acest motiv, valoarea returnată trebuie salvată într-o variabilă de tip size_type; utilizarea unei variabile de tip int sau unsigned nu este sigură (de exemplu, este posibil ca sizeof(unsigned long) să difere de sizeof(size_type), şi în consecință (unsigned long)-1 diferă de (size_type)-1).

```
int i = s.find("abc");                  //presupunem că returnează npos
if (i == string::npos) {                //compararea poate eşua! posibilă eroare
        ...
}
size_type ii = s.find("abc");           //corect
if (ii == string::npos) {               //OK
        ...
}
```

9. Compararea obiectelor string

Operatorii uzuali (<, >, <=, >=, ==, !=) au fost supraîncărcați astfel încât să permită compararea lexicografică, conform clasei caracteristice curente, a caracterelor din două obiecte string sau dintr-un string şi un C-şir; operatorii returnează valori bool.

Pentru a compara subşiruri, clasa *string* pune la dispoziție metoda compare(), supraîncărcată astfel:

int string::compare(const string& s) const	Compară caracterele din *this cu cele din s.
int string::compare(size_type i, size_type n, const string& s) const	Compară cel mult n caractere din *this, începând de la poziția i, cu caracterele din s.
int string::compare(size_type i, size_type n, const string& s, size_type si, size_type sn) const	Compară cel mult n caractere din *this, începând de la poziția i, cu cel mult si caractere din s, începând de la poziția sn.
int string::compare(const char* s) const	Compară caracterele din *this cu cele din C-şirul s.
int string::compare(size_type i, size_type n, const char* s) const	Compară cel mult n caractere din *this, începând de la poziția i, cu caracterele din C-şirul s.
int string::compare(size_type i, size_type n, const char* s, size_type sn) const	Compară cel mult n caractere din *this, începând de la poziția i, cu sn caractere din tabloul s; '\0' nu are semnificație specială.

10. I/O

ostream& operator<< (ostream& f, const string& s)	Scrie în fluxul f caracterele string-ului s. Dacă f.width()>0, scrie în f cel puțin f.width() caractere şi setează f.width() la 0.
istream& operator>>(istream& f, string& s)	Transferă în s caracterele din fluxul f până când: • urmează un caracter whitespace • f nu se mai află într-o stare validă (de ex, EOF) • f.width()>0 şi s-au citit f.width() caractere • s-au citit max_size() caractere Caracterele whitespace de la inceput sunt ignorate dacă pentru f este setat flag-ul *skipws*.
istream& getline(istream& f, string& s)	Transferă în s toate caracterele (inclusiv whitespace) din fluxul f până la întâlnirea delimitatorului de linie sau EOF. Delimitatorul de linie este extras din f dar nu este inserat în s.
istream& getline(istream& f, string& s, char d)	Analog, cu precizarea explicită a unui caracter delimitator de linie.

11. Exemplu

Se prezintă ca exemplu un program care citeşte linie cu linie conţinutul unui flux de intrare şi, pentru fiecare linie, afişează cuvintele oglindite.

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

const string motto = "Trei inele pentru stapanii elfi cei de sub soare,"
        "Sapte pentru ei, piticii de vita din sali de stanca,"
        "Noua, Oamenilor care stiu ca-n lumea lor se moare,"
        "Unul pentru el, Seniorul Intunecimii-n noaptea lui adanca"
        "Unde-s Umbrele in Tinutul Mordor, ca sa le gaseasca."
        "Si pe toate sa le-adune un inel, si altul nime,"
        "Sa le ferece pe toate, astfel sa le stapaneasca,"
        "Unde-s Umbrele, in Tinutul Mordor, in intunecime.";

const string delimitatori(" \t,.;");

int main(void){

istringstream is(motto);
string linie;
string::size_type begin, end;

while (getline(is, linie)){                                     //pentru fiecare linie citită
        begin = linie.find_first_not_of(delimitatori);          //începutul cuvântului
        while (begin != string::npos){
                end = linie.find_first_of(delimitatori, begin); //sfârşitul cuvântului
                if (end == string::npos)
                        end = linie.length();                   //sfârşitul cuvântului coincide cu sfârşitul liniei
                for (int i = (end-1); i >= static_cast<int>(begin); --i) //caracterele sunt afişate în ordine inversă
                        cout << linie[i];
                cout << ' ';
                begin = linie.find_first_not_of(delimitatori, end);      //începutul următorului cuvânt
        }
        cout << linie;
}
return 0;
}
```

Dacă nu se efectuează conversia static_cast<int> (begin), programul poate executa o buclă infinită sau se poate termina cu eroare la execuţie. Aceasta deoarece:

- begin este de tip intreg fără semn;
- la compararea lui begin cu i, acesta din urmă este convertit automat spre un întreg fără semn;
- orice întreg fără semn este >= 0;
- după primul apel al lui find_first_not_of() begin are valoarea 0!

Bibliografie

1. Bjarne Stroustrup – The C++ Programming Language, Third Edition, 1997
2. Nicolai M. Josuttis – The C++ Standard Library. A Tutorial and Reference, 1999
3. Bruce Eckel – Thinking In C++, 1999