Αλφαριθμητικά

Αρχικοποίηση

- Template class **`basic_string`**
  - Μεταχείριση αλφαριθμητικών (αντιγραφή, αναζήτηση κτλ.)
    - **`typedef basic_string< char > string;`**
    - Also **`typedef`** for **`wchar_t`** (**`std::wstring`**)
  - Βιβλιοθήκη: **`<string>`**
- Αρχικοποίηση **`string`**
  - **`string s1("word");`** `word`
  - **`string s2( 5, 'x' );`** `xxxxx`
  - **`string s3 = "word"`** `word`
    - Έμμεσα καλεί τον δημιουργό

Αρχικοποίηση

- Δεν επιτρέπεται μετατροπή από **`int`** ή **`char`**
  - Τα παρακάτω προκαλούν σφάλματα:
    - **`string err1 = 'd';`**
    - **`string err2('x' );`**
    - **`string err3 = 45;`**
    - **`string err4( 5 );`**
  - Μπορούμε να αναθέσουμε έναν χαρακτήρα με :
    - **`s = 'n';`**
    - **`std::string::operator=`** overloaded for char

Χαρακτηριστικά

- **`string`**
  - Δεν είναι απαραίτητο να τερματίζονται με **`null`**
  - Η μέθοδος **`length`** επιστρέφει τον αριθμό χαρακτήρων: `s1.length()`
  - Ο τελεστής **`[]`** επιτρέπει προσπέλαση χαρακτήρων: **`s1[0]`**
  - Ένα **`string`** δεν είναι δείκτης
  - Ανάκτηση από κανάλι
    - `cin >> stringObject;`
    - `getline( cin, s)`
      - Διαβάζει μέχρι τον χαρακτήρα επόμενης γραμμής.

Ανάθεση

- Ανάθεση
  - `s2 = s1;` ή `s2.assign(s1);`
    - Δημιουργεί αντίγραφο
  - `myString.assign(s, start, N);`
    - αντιγράφει `N` χαρακτήρες από το string `s`, ξεκινώντας από τη θέση `start`
  - Ανάθεση συγκειμένων θέσεων (χαρακτήρων)
    - `s1[0] = s2[4];`

Μέθοδοι at και append

- Έλεγχος εμβέλειας
  - `s.at( index );`
    - Επιστρέφει τον χαρακτήρα στην θέση `index`
    - Μπορεί να πετάξει εξαίρεση `out_of_range` αν δοθεί αριθμός εκτός ορίων
  - Ο τελεστής `[]` δεν κάνει τέτοιο έλεγχο
- Προσθήκη στο τέλος (Concatenation)
  - `s.append("winter");` ή `s += "winter";`
    - Προσθέτουν το `"pet"` στο τέλος του `s`
  - `s.append( ssrc, start, N );`
    - Τοποθετεί στο τέλος του `s`, `N` χαρακτήρες από τον `ssrc`, ξεκινώντας από την θέση `start`

fig15_01.cpp (1 of 3)

```
1  // Fig. 15.1: fig15_01.cpp
2  // Demonstrating string assignment and concatenation.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <string>
9
10 using std::string;
11
12 int main()
13 {
14    string string1( "cat" );
15    string string2;
16    string string3;
17
18    string2 = string1;          // assign string1 t
19    string3.assign( string1 );  // assign string1 to string3
20    cout << "string1: " << string1 << "\nstring2: " << string2
21         << "\nstring3: " << string3 << "\n\n";
22
```

String initialization and assignment.

```
Output
string1: cat
string2: cat
string3: cat
```


fig15_01.cpp (2 of 3)

```
23    // modify string2 and string3
24    string2[ 0 ] = string3[ 2 ] = 'r';
25
26    cout << "After modification of string2 and string3:\n"
27         << "string1: " << string1 << "\nstring2: " << string2
28         << "\nstring3: ";
29
30    // demonstrating member function at
31    for ( int i = 0; i < string3.length();
32       cout << string3.at( i );
33
34    // declare string4 and string5
35    string string4( string1 +
36    string string5;
37
38    // overloaded +=
39    string3 += "pet";           // create "carpet"
40    string1.append( "acomb" );  // create "catacomb"
41
42    // append subscript locations 4 through end of string1
43    // create string "comb" (string5 was initially empty)
44    string5.append( string1, 4, string1.length() );
45
46    cout << "\n\nAfter concatenation:\nstring1: " << string1
47         << "\nstring2: " << string2 << "\nstring3: "
48         << string3 << "\nstring4: " << string4
49         << "\nstring5: " << string5 << endl;
50
```

```
After modification of string2 and string3:
string1: cat
string2: rat
string3: car
```

Note use of member function **`at`** instead of **`[]`**.

```
After concatenation:
string1: catacomb
string2: rat
string3: carpet
string4: catapult
string5: comb
```

```
51        return 0;
52
53    }  // end main
```

```
string1: cat
string2: cat
string3: cat

After modification of string2 and string3:
string1: cat
string2: rat
string3: car

After concatenation:
string1: catacomb
string2: rat
string3: carpet
string4: catapult
string5: comb
```

fig15_01.cpp (3 of 3)

fig15_01.cpp output (1 of 1)

Σύγκριση

- Υπερφορτωμένοι τελεστές σύγκρισης:
  - **==**, **!=**, **<**, **>**, **<=** και **>=**
  - Επιστρέφουν τιμή **bool**
- **s1.compare(s2)**
  - Επιστρέφει θετικό αν το **s1** είναι μεγαλύτερο λεξικογραφικά
    - Σύγκριση γράμμα προς γράμμα
    - Αρνητικό αν είναι μικρότερο, 0 αν είναι ίσα
  - **s1.compare(start, length, s2, start, length)**
    - Συγκρίνει επιμέρους μέρη του **s1** και **s2**
  - **s1.compare(start, length, s2)**
    - Συγκρίνει επιμέρους μέρος του **s1** με ολόκληρο το **s2**

Μέθοδοι substr , swap

- Η **substr** επιστρέφει ένα μέρος του αλφαριθμητικού
  - **s.substr(start, N);**
  - Επιστρέφει ένα string παίρνοντας **N** χαρακτήρες του **s**, ξεκινώντας από την θέση **start**
- Μέθοδος **swap**
  - **s1.swap(s2);**
  - Ανταλλάσσει τα περιεχόμενα των δύο strings

Χρήσιμες Μεθόδοι

- **s1.size()** και **s1.length()**
  - Επιστρέφει το μέγεθος του string
- **s1.capacity()**
  - Επιστρέφει πόσα στοιχεία μπορούν να αποθηκευτούν χωρίς να γίνει reallocation
- **s1.max_size()**
  - Επιστρέφει το μέγιστο μέγεθος του string
- **s1.empty()**
  - Επιστρέφει true αν είναι άδειο
- **s1.resize(newlength)**
  - Αλλάζει το μέγεθος σε newlength

fig15_05.cpp
(1 of 3)

```
1  // Fig. 15.5: fig15_05.cpp
2  // Demonstrating member functions related to size and capacity.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7  using std::cin;
8  using std::boolalpha;
9
10 #include <string>
11
12 using std::string;
13
14 void printStatistics( const string & );
15
16 int main()
17 {
18    string string1;
19
20    cout << "Statistics before input:\n" << boolalpha;
21    printStatistics( string1 );
22
23    // read in "tomato"
24    cout << "\n\nEnter a string: ";
25    cin >> string1;  // delimited by whitespace
26    cout << "The string entered was: " << string1;
```


fig15_05.cpp
(2 of 3)

```
27
28    cout << "\nStatistics after input:\n";
29    printStatistics( string1 );
30
31    // read in "soup"
32    cout << "\n\nThe remaining string is: " << string1 << endl;
33    printStatistics( string1 );
34
35    // append 46 characters to string1
36    string1 += "1234567890abcdefghijklmnopqrstuvwxyz1234567890";
37    cout << "\n\nstring1 is now: " << string1 << en
38    printStatistics( string1 );
39
40    // add 10 elements to string1
41    string1.resize( string1.length() + 10 );
42    cout << "\n\nStats after resizing by (length + 10):\n";
43    printStatistics( string1 );
44
45    cout << endl;
46    return 0;
47
48 }  // end main
49
50
```

Resize string.

```
51 // display string statistics
52 void printStatistics( const string &stringRef )
53 {
54    cout << "capacity: " << stringRef.capacity()
55         << "\nmax size: " << stringRef.max_size()
56         << "\nsize: " << stringRef.size()
57         << "\nlength: " << stringRef.length()
58         << "\nempty: " << stringRef.empty();
59
60 }  // end printStatistics
```

Display various string characteristics.

fig15_05.cpp (3 of 3)

fig15_05.cpp output (1 of 2)

```
Statistics before input:
capacity: 0
max size: 4294967293
size: 0
length: 0
empty: true

Enter a string: tomato soup
The string entered was: tomato
Statistics after input:
capacity: 31
max size: 4294967293
size: 6
length: 6
empty: false
```


fig15_05.cpp output (2 of 2)

```
The remaining string is: soup
capacity: 31
max size: 4294967293
size: 4
length: 4
empty: false

string1 is now: soup1234567890abcdefghijklmnopqrstuvwxyz1234567890
capacity: 63
max size: 4294967293
size: 50
length: 50
empty: false

Stats after resizing by (length + 10):
capacity: 63
max size: 4294967293
size: 60
length: 60
empty: false
```

Μέθοδοι αναζήτησης

- Επιστρέφουν την θέση που βρέθηκε
- Αν δεν βρεθεί επιστρέφουν, **`string::npos`** (σταθερά)
  - **`s1.find( s2 )`**
  - **`s1.rfind( s2 )`**
    - Ξεκινάει την αναζήτηση από δεξιά
  - **`s1.find_first_of( s2 )`**
    - Επιστρέφει την πρώτη εμφάνιση στο **s1** οποιουδήποτε χαρακτήρα του **`s2`**
  - **`s1.find_last_of( s2 )`**
    - Επιστρέφει την τελευταία εμφάνιση στο **s1** οποιουδήποτε χαρακτήρα του **`s2`**
  - **`s1.find_first_not_of( s2 )`**
    - Επιστρέφει την πρώτη εμφάνιση στο **s1** οποιουδήποτε χαρακτήρα που δεν υπάρχει στο **`s2`**
  - **`s1.find_last_not_of( s2 )`**
    - Επιστρέφει την τελευταία εμφάνιση στο **s1** οποιουδήποτε χαρακτήρα που δεν υπάρχει στο **`s2`**

Αφαίρεση - Αντικατάσταση

- **`s1.erase( start )`**
  - Αφαίρεση από τη θέση **`start`** μέχρι το τέλος
- **`s1.replace( begin, N, s2)`**
  - Ξεκινώντας από την θέση **begin** του **s1** αντικατέστησε τους επόμενους **N** χαρακτήρες, με χαρακτήρες από το **s2**
- **`s1.replace( begin, N, s2, index, num )`**
  - Όμοια με παραπάνω αλλά καθορίζεται επιπλέον από ποιο σημείο index του s2 θα ξεκινήσουμε και πόσους χαρακτήρες θα χρησιμοποιήσουμε
- Η θέση **`string::npos`** αναπαριστά το μέγιστο μέγεθος του string

Outline

fig15_07.cpp (1 of 2)

```
1  // Fig. 15.7: fig15_07.cpp
2  // Demonstrating string member functions erase and replace.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <string>
9
10 using std::string;
11
12 int main()
13 {
14    // compiler concatenates all parts into one string
15    string string1( "The values in any left subtree"
16       "\nare less than the value in the"
17       "\nparent node and the values in"
18       "\nany right subtree are greater"
19       "\nthan the value in the parent node" );
20
21    cout << "Original string:\n" << string1 << endl << endl;
22
23    // remove all characters from (and including) location 62
24    // through the end of string1
25    string1.erase( 62 );
26
```


Outline

fig15_07.cpp

```
27    // output new string
28    cout << "Original string after er
29         << "\n\nAfter first replacer
30
31    // replace all spaces with period
32    int position = string1.find( " " );
33
34    while ( position != string::npos ) {
35       string1.replace( position, 1, "." );
36       position = string1.find( " ", position + 1 );
37    } // end while
38
39    cout << string1 << "\n\nAfter second replacemer
40
41    // replace all periods with two semicolons
42    // NOTE: this will overwrite characters
43    position = string1.find( "." );
44
45    while ( position != string::npos ) {
46       string1.replace( position, 2, "xxxxx;;yyy", 5, 2 );
47       position = string1.find( ".", position + 1 );
48    } // end while
49
50    cout << string1 << endl;
51    return 0;
52
53 } // end main
```

`string::npos` represents max string length.

Find each space and replace with a `'.'`

Start each search at the next position.

Replace all `'.'` with two semicolons (the two characters at index 5).

```
Original string:
The values in any left subtree
are less than the value in the
parent node and the values in
any right subtree are greater
than the value in the parent node

Original string after erase:
The values in any left subtree
are less than the value in the


After first replacement:
The.values.in.any.left.subtree
are.less.than.the.value.in.the


After second replacement:
The;;alues;;n;;ny;;eft;;ubtree
are;;ess;;han;;he;;alue;;n;;he
```

fig15_07.cpp
output (1 of 1)

Ένθεση

- **`s1.insert( index, s2 )`**
  - Ενθέτει το **`s2`** πριν από τη θέση **`index`**
- **`s1.insert( index, s2, index2, N );`**
  - Ενθέτει ένα μέρος του **`s2`** **(N** χαρακτήρες ξεκινώντας από τη θέση **index2)** πριν από τη θέση **`index`**

Μετατροπή

- Μέθοδοι μετατροπής:
  - **`s1.copy( ptr, N, index )`**
    - αντιγράφει **`N`** χαρακτήρες στον πίνακα **`ptr (*char)`**
    - Ξεκινάει από την θέση **`index`**
    - Πρέπει να τερματιστεί με null
  - **`s1.c_str()`**
    - Επιστρέφει **`const char *`**
    - Δεν χρειάζεται να τερματιστεί με null
  - **`s1.data()`**
    - Επιστρέφει **`const char *`**
    - Πρέπει να τερματιστεί με null

Iterators

- Iterators
  - Διαπέραση χαρακτήρων (από αρχή ή από το το τέλος)
  - Πρόσβαση στους επιμέρους χαρακτήρες
  - Όμοια με δείκτες

- Βασική Χρήση
  - Δημιουργία
    - **`string::const_iterator i = s.begin();`**
    - **`const`**, δεν επιτρέπεται τροποποίηση
  - Αναφορά
    - **`*i;  // αναφορά στον χαρακτήρα`**
    - **`++i;  // μεταφορά στον επόμενο χαρακτήρα`**
  - Έλεγχος για τέλος
    - **`i != s.end()`**
    - **`end`** επιστρέφει iterator μετά το τελευταίο στοιχείο του **`s`**

```
1  // Fig. 15.10: fig15_10.cpp
2  // Using an iterator to output a string.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <string>
9
10 using std::string;
11
12 int main()
13 {
14    string string1( "Testing iterators" );
15    string::const_iterator iterator1 = string1.begin();
16
17    cout << "string1 = " << string1
18         << "\n(Using iterator iterator1) strin
19
20    // iterate through string
21    while ( iterator1 != string1.end() ) {
22       cout << *iterator1;  // dereference iterator to get char
23       ++iterator1;  // advance iterator to next char
24    } // end while
25
26    cout << endl;
27    return 0;
28
29 }  // end main
```

Print each character using the iterator.

Outline

fig15_10.cpp (1 of 1)


```
string1 = Testing iterators
(Using iterator iterator1) string1 is: Testing iterators
```

Outline

fig15_10.cpp output (1 of 1)

String Streams

- I/O of strings to and from memory
  - Called in-memory I/O or string stream processing
  - Classes
    - **istringstream** (είσοδος από string)
    - **ostringstream** (έξοδος σε string)
    - **<sstream> <iostream>**
- Κανάλι εξόδου ostringstream
  - `ostringstream outputString;` Δημιουργία string-καναλιού
  - `outputString << s1 << s2;` Αποθήκευση strings στο string-κανάλι
  - Μέθοδος : **str**
    - Επιστρέφει ως **string** το περιεχόμενο του string-καναλιού
    - `outputString.str()`
- Κανάλι εισόδου istringstream
  - `istringstream inputString ( myString );` Δημιουργία string-καναλιού από string
  - `inputString >> string1 >> string2` Ανάκτηση από το string-κανάλι σε strings
  - Με παρόμοιο τρόπο που διαβάζουμε από την κύρια είσοδο, **cin**

Outline

fig15_11.cpp (1 of 2)

```
1  // Fig. 15.11: fig15_11.cpp
2  // Using a dynamically allocated ostringstream object.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <string>
9
10 using std::string;
11
12 #include <sstream>
13
14 using std::ostringstream;
15
16 int main()
17 {
18    ostringstream outputString;  // create ostringstream instance
19
20    string string1( "Output of several data types " );
21    string string2( "to an ostringstream object:" );
22    string string3( "\n        double: " );
23    string string4( "\n           int: " );
24    string string5( "\naddress of int: " );
```

Create **ostringstream** object.

```
25
26    double double1 = 123.4567;
27    int integer = 22;
28
29    // output strings, double and int to outputString
30    outputString << string1 << string2 << string3 << double1
31                 << string4 << integer << string5 << &integer;
32
33    // call str to output contents
34    cout << "outputString contains:\n" << outputString.str();
35
36    // add additional characters and call str to output string
37    outputString << "\nmore c
38    cout << "\n\nafter additi
39         << "outputString con                    ()
40         << endl;
41
42    return 0;
43
44 }  // end main
```

Output format just like to writing to **cout**.

Outline

fig15_11.cpp (2 of 2)


```
outputString contains:
Output of several data types to an ostringstream object:
        double: 123.457
           int: 22
address of int: 0012FE94

after additional stream insertions,
outputString contains:
Output of several data types to an ostringstream object:
        double: 123.457
           int: 22
address of int: 0012FE94
more characters added
```

Outline

fig15_11.cpp output (1 of 1)

fig15_12.cpp (1 of 2)

```
1  // Fig. 15.12: fig15_12.cpp
2  // Demonstrating input from an istringstream object.
3  #include <iostream>
4
5  using std::cout;
6  using std::endl;
7
8  #include <string>
9
10 using std::string;
11
12 #include <sstream>
13
14 using std::istringstream;
15
16 int main()
17 {
18    string input( "Input test 123 4.7 A" );
19    istringstream inputString( input );
20    string string1;
21    string string2;
22    int integer;
23    double double1;
24    char character;
25
```

Create and initialize **istringstream** object.

fig15_12.cpp (2 of 2)

```
26    inputString >> string1 >> string2 >> integer >> double1
27               >> character;
28
29    cout << "The following items were extracted\n"
30         << "from the istringstream object:"
31         << "\nstring: " << string1
32         << "\nstring: " << string2
33         << "\n   int: " << integer
34         << "\ndouble: " << double1
35         << "\n  char: " << character;
36
37    // attempt to read from empty stream
38    long value;
39
40    inputString >> value;
41
42    // test stream results
43    if ( inputString.good() )
44       cout << "\n\nlong value is: " << value << endl;
45    else
46       cout << "\n\ninputString is empty" << endl;
47
48    return 0;
49
50 }  // end main
```

Read data into variables.

good returns 1 if can still read data (no EOF, bad bits, etc). In this case, there is no data, so the test fails.

```
The following items were extracted
from the istringstream object:
string: Input
string: test
   int: 123
double: 4.7
  char: A

inputString is empty
```

fig15_12.cpp
output (1 of 1)