Chapter 3

Simple Data Types and Basic Support Operations

This section describes simple data types like strings, streams and gives some information about error handling, memory management and file system access. The stream data types described in this section are all derived from the C++ stream types *istream* and *ostream*. They can be used in any program that includes the <LEDA/stream.h> header file. Some of these types may be obsolete in combination with the latest versions of the standard C++ I/O library.

3.1 Strings (string)

1. Definition

An instance s of the data type *string* is a sequence of characters (type *char*). The number of characters in the sequence is called the length of s. A string of length zero is called the empty string. Strings can be used wherever a C++ *const char** string can be used.

Strings differ from the C++ type *char** in several aspects: parameter passing by value and assignment works properly (i.e., the value is passed or assigned and not a pointer to the value) and *strings* offer many additional operations.

#include < LEDA/string.h >

2. Types

string::*size_type*	the size type.

3. Creation

string s;	introduces a variable s of type *string*. s is initialized with the empty string.

string $s(const\ char * p)$; introduces a variable s of type *string*. s is initialized with a copy of the C++ string p.

string $s(char\ c)$; introduces a variable s of type *string*. s is initialized with the one-character string "c".

string $s(const\ char * format, \ldots)$;
introduces a variable s of type *string*. s is initialized with the string produced by printf($format$,...).

4. Operations

int	s.length( )	returns the length of string s.
char&	$s[int\ i]$	returns the character at position i. *Precondition*: $0 \leq i \leq s$.length()-1.
string	$s(int\ i, int\ j)$	returns the substring of s starting at position $\max(0, i)$ and ending at position $\min(j, s.\text{length}()-1)$. If $\min(j, s.\text{length}()-1) < \max(0, i)$ then the empty string is returned.
string	s.head($int\ i$)	returns the first i characters of s.
string	s.tail($int\ i$)	returns the last i characters of s.
int	s.pos($string\ s1$, $int\ i$)	returns the minimum j such that $j \geq i$ and s_1 is a substring of s starting at position j (returns -1 if no such j exists).
int	s.pos($const\ string\&\ s1$)	returns $\text{pos}(s1, 0)$.
bool	s.contains($const\ string\&\ s1$) true iff $s1$ is a substring of s.	
string	s.insert($int\ i$, $string\ s1$)	returns $s(0, i-1) + s_1 + s(i, s.\text{length}()-1)$.
string	s.replace($const\ string\&\ s1$, $const\ string\&\ s2$, $int\ i = 1$) returns the string created from s by replacing the i-th occurrence of s_1 in s by s_2. Remark: The occurences of s_1 in s are counted in a non-overlapping manner, for instance the string $sasas$ contains only one occurence of the string sas.	
string	s.replace($int\ i$, $int\ j$, $const\ string\&\ s1$) returns the string created from s by replacing $s(i, j)$ by s_1. *Precondition*: $i \leq j$.	

string *s*.replace(*int i*, *const string*& *s1*)

returns the string created from s by replacing $s[i]$ by s_1.

string *s*.replace_all(*const string*& *s1*, *const string*& *s2*)

returns the string created from s by replacing all occurrences of s_1 in s by s_2.
Precondition: The occurrences of s_1 in s do not overlap (it's hard to say what the function returns if the precondition is violated.).

string *s*.del(*const string*& *s1*, *int i* = 1)

returns s.replace(s_1,"",i).

string *s*.del(*int i*, *int j*) returns s.replace(i,j,"").

string *s*.del(*int i*) returns s.replace(i,"").

string *s*.del_all(*const string*& *s1*)

returns s.replace_all(s_1,"").

void *s*.read(*istream*& *I*, *char delim* = ' ')

reads characters from input stream I into s until the first occurrence of character *delim*.

void *s*.read(*char delim* = ' ') same as s.read(*cin*,*delim*).

void *s*.read_line(*istream*& *I*) same as s.read(I,'\ n').

void *s*.read_line( ) same as s.read_line(*cin*).

void *s*.read_file(*istream*& *I*) same as s.read(I,'EOF').

void *s*.read_file( ) same as s.read_file(*cin*).

string& s += *const string*& x appends x to s and returns a reference to s.

string *const string*& x + *const string*& y

returns the concatenation of x and y.

bool *const string*& x == *const string*& y

true iff x and y are equal.

bool *const string*& x != *const string*& y

true iff x and y are not equal.

bool *const string*& x < *const string*& y

true iff x is lexicographically smaller than y.

bool *const string& x > const string& y*

true iff x is lexicographically greater than y.

bool *const string& x ≤ const string& y*

returns $(x < y) \mid (x == y)$.

bool *const string& x ≥ const string& y*

returns $(x > y) \mid (x == y)$.

istream& *istream& I ≫ string& s*

same as s.read(I,' ').

ostream& *ostream& O ≪ const string& s*

writes string s to the output stream O.

5. Implementation

Strings are implemented by C++ character vectors. All operations involving the search for a pattern $s1$ in a string s take time $O(s.lenght() * s1.length())$, [ ] takes constant time and all other operations on a string s take time $O(s.length())$.