La Libreria Standard del C++

Gestione di Stringhe

Gestione di Stringhe

- La gestione delle stringhe in C è di basso livello: si basa su
  - puntatore a carattere
  - uso delle funzioni della libreria C `string.h`
- Nasconde numerosi trabocchetti nei quali possono incorrere anche programmatori esperti

Le stringhe in C

- Definizione:

```
char * s;
s = new char[12];
char *s1="ingegneria";
```

- Attenzione alle dimensioni: il carattere `\0` indica dove termina la stringa
- Quindi la dimensione di s1 è pari a 11 caratteri

La libreria C `string.h`

- Copiare una stringa in un'altra
  `char* strcpy(char *, char*)`
- Concatenare due stringhe
  `char* strcat(char*, char *)`
- Confrontare due stringhe
  `int strcmp(char *, char*)`
- Dimensione di una stringa
  `int strlen(char *)`

La Classe C++ `string`

- Definita nella libreria Standard del C++
- Fornisce metodi per:
  - Inizializzazione
  - Copia, concatenazione, etc.
  - Accesso ai singoli caratteri
  - Confronto
  - Ricerche
- NB: lo standard è recente (1998), quindi poco diffuso

La Classe `string`

- Una stringa C++ è una sequenza di caratteri che possono essere indicizzati.
- (Come per i `vector`) la capacità di un oggetto di tipo `string` può crescere dinamicamente.
- Se il numero di caratteri da memorizzare in una stringa eccede la capacità iniziale della stringa, la memoria necessaria al nuovo valore viene allocata dinamicamente.
- Tutto questo avviene in maniera trasparente, senza richiedere nessun intervento da parte del programmatore.

La classe `string`

- I programmi che usano la classe string devono includere:
  - l'header file `string`
  - il namespace `std`

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

Definizione e Costruttori

```
string s1;
string s2 ("una stringa");
string s3 = "valore iniziale";
```

- Il **costruttore di copia** permette di definire una stringa inizializzandone il valore con quello di una stringa precedentemente definita:

```
string s4 (s3);
```

- In questi esempi la **capacità** (vedi oltre) iniziale è esattamente pari al numero di caratteri del valore di inizizalizzazione.
- Un costruttore alternativo permette di impostare la capacità e di inizializzare la stringa con una copia ripetuta di un singolo carattere:

```
string s7 (3, 'a'); // equivalente a string s7 ("aaa")
```

Dimensione e Capacità

- (Come per Vector), la dimensione di un oggetto di tipo `string` può essere verificata con i metodi `size()` e `length()`(sono sinonimi); per conoscere la capacità si usa il metodo `capacity()`.

```
cout << s6.size() << endl;
cout << s6.capacity() << endl;
```

- Il metodo `reserve()` permette di reimpostare la capacità di un oggetto string. Il metodo `max_size()` ritorna la massima dimensione disponibile per di una stringa:

```
s6.reserve(200);// cambia la capacità a 200
cout << s6.capacity() << endl;
cout << s6.max_size() << endl;
```

Dimensione e Capacità

- Il metodo `resize()` cambia la dimensione di una stringa, o troncandola o inserendo nuovi caratteri. Il secondo argomento di questo metodo può essere usato per specificare il carattere che deve essere inserito nelle nuove posizioni.

```
s7.resize(15, '\t');
  //aggiunge 15 tab alla fine
cout << s7.length() << endl;
 // ora la dimensione e' 15
```

- Il metodo `empty()` ritorna `true` se la stringa è vuota (è più efficiente rispetto a `length()==0`)

```
if (s7.empty())
  cout << "stringa vuota" << endl;
```

Assegnare, concatenare, scambiare

- Ad un oggetto di tipo `string` può essere assegnato una stringa costante, o un singolo carattere:

```
s1 = s2;
s2 = "a new value";
s3 = ‘a’;
```

- L’operatore += permette di gestire la concatenazione di stringhe (l’operando di destra viene concatenato alla fine dell’oggetto che invoca il metodo)

```
s3 += “bc";  // ora in s3 c’e’ abc
```

Assegnare, Concatenare, Scambiare

- I metodi `assign()` e `append()` permettono di concatenare un sottoinsieme dell’operando di destra.
- Due argomenti, `pos` e `n`, indicano che gli `n` valori a partire da `pos` devono essere assegnati o concatenati all’oggetto invocante:

```
s4.assign(s2, 0, 3);
  // assegna i primi 3 caratteri di s2 a s4

s4.append(s5, 2, 3);
  // concatena a s4 i caratteri dalla posizione 2
  // alla posizione 2+3 di s5
```

Concatenare e Scambiare

- L'operatore somma + è usato per concatenare due stringhe: crea una copia dell'argomento di sinistra e gli appende l'argomento di destra:

```
cout << (s2 + s3) << endl;
```

- (Come per tutti i contenitori della Standard C++ Library), i contenuti di due stringhe possono essere scambiati usando il metodo `swap()`:

```
s5.swap (s4); // scambia s4 con s5
```

Accedere ai caratteri

- L'operatore di indicizzazione (subscript) e il metodo `at()` permettono di accedere ai singoli caratteri di una stringa. La differenza tra i due è che `at()` lancia un'eccezione se si cerca di accedere ad una locazione maggiore o uguale di `size()`.

```
cout << s4[2] << endl;
s4[2] = 'X';

cout << s4.at(2) << endl;
S4.at(2) = 'X'
```

Inserire, cancellare, sostituire

- I metodi `insert()` ed `erase()` consentono l'inserimento e la rimozione dei caratteri compresi in un intervallo specificato dagli argomenti.
- Il metodo `replace()` consente di sostituire i caratteri nell'intervallo specificato con nuovi valori.

```
s3.insert (3, "abc");
  // inser. abc dopo la posizione 3

s3.erase (4, 2);
  // rimuove le posizioni 4 e 5

s3.replace (4, 2, "pqr");
  // sostituisce le posizioni da 4 a 4+2-1 con pqr
```

Copia e Sottostringhe

- Il metodo `copy()` genera una sottostringa e la assegna al primo argomento. L'intervallo di valori per delimitare la sottostringa può essere specificato dalla posizione iniziale, o da posizione e lunghezza dell'intervallo

```
s3.copy (s4, 2);
// assegna a s4 i caratteri da 2 alla fine di s3
s5.copy (s4, 2, 3);
// assegna a s4 i caratteri da 2 a 4 of s5
```

- Il metodo `substr()` ritorna una sottostringa dell'oggetto invocante. :

```
cout << s4.substr(3) << endl;  // stampa da 3 alla fine
cout << s4.substr(3, 2) << endl;
  //stampa due caratteri a partire dalla posizione 3
```

Ricerche

- Il metodo `find()` determina la prima occorrenza dell'argomento (di tipo `string`) nella stringa corrente.
- Un argomento opzionale (intero) permette di specificare la posizione da dove iniziare la ricerca.
- Se la ricerca ha esito positivo, il metodo ritorna la posizione iniziale della sottostringa cercata; altrimenti ritorna un valore maggiore della dimensione della stringa.
- Il metodo `rfind()` è simile, ma comincia dalla fine.

```
s1 = "mississippi";
cout << s1.find("ss") << endl; // stampa 2
cout << s1.find("ss", 3) << endl; // stampa 5
cout << s1.rfind("ss") << endl; // stampa 5
cout << s1.rfind("ss", 4) << endl; // stampa 2
```