# Programmazione di sistema

### Anno accademico 2018-2019

## Esercitazione 1

Si progetti e si realizzi la classe StringBuffer che permetta l'accesso ad un buffer di caratteri: tale classe costituisce un contenitore dinamico in grado di memorizzare stringhe di caratteri di diversa lunghezza. A tale scopo incapsula un puntatore nativo ed una coppia di interi: il primo indica la dimensione totale del buffer, il secondo quanti caratteri sono attualmente in uso.

La classe offre le seguenti funzioni membro pubbliche:

- StringBuffer() – costruisce un oggetto con un buffer di dimensione iniziale ragionevole, inizializzandolo con il terminatore di stringa
- StringBuffer(const char * **str** ) – costruisce un oggetto allocando un buffer di dimensione sufficiente a contenere l'array di caratteri str (compreso il terminatore finale) e segnando correttamente il numero di caratteri effettivamente utilizzati
- StringBuffer(const StringBuffer& **sb** ) – costruisce un oggetto con un buffer iniziale di dimensione adeguata a contenere i caratteri contenuti nell'oggetto sb e inizializzato con una copia di tali caratteri
- ~StringBuffer() – rilascia le risorse contenute nell'oggetto
- StringBuffer& operator= (const StringBuffer& **sb** ) assegna, al buffer corrente, il contenuto del buffer sb
- size_t size () – restituisce il numero di caratteri utilizzati dalla stringa memorizzata nel buffer;
- size_t capacity () – restituisce la dimensione totale del buffer di caratteri attualmente allocato;
- void clear () – porta a 0 il numero di caratteri utilizzati;
- void insert (const char* str , size_t pos ) – inserisce il contenuto s nella posizione pos del buffer; o se pos è maggiore di size (), inserisce spazi tra size () e pos, riallocando il buffer se necessario
- void insert (const StringBuffer& sb , size_t pos ) – inserisce il contenuto di sb nella posizione pos del buffer; o se pos è maggiore di size (), inserisce spazi tra size () e pos, riallocando il buffer se necessario
- void append (const char* str ) – aggiunge i caratteri contenuti in str in coda a quelli memorizzati nel buffer, riallocando il buffer se necessario;
- void append (const StringBuffer& sb ) – aggiunge i caratteri contenuti nell'oggetto sb in coda a quelli memorizzati nel buffer, riallocando il buffer se necessario;
- cost char* c_str () – restituisce un puntatore in sola lettura al buffer interno opportunamente terminato con un "\0";
- void set (const char* str ) – sostituisce la stringa memorizzata nel buffer con il contenuto dell'array s , riallocando il buffer se necessario;

- void set (const StringBuffer& s ) – sostituisce la stringa memorizzata nel buffer con il contenuto dell'oggetto s, riallocando il buffer se necessario.

Si faccia attenzione a rilasciare correttamente tutta la memoria allocata e ad allocare dinamicamente la memoria aggiuntiva eventualmente necessaria qualora la stringa da memorizzare richiedesse un buffer di dimensioni maggiori, tenendo conto che deve esserci spazio nel buffer anche per il terminatore della stringa ("\0").

## Nota bene

Per lo svolgimento dell'esercitazione è **necessario utilizzare i puntatori nativi** . Nelle prossime esercitazioni, invece, si useranno contenitori standard e smart pointer.

## Esempio di uso

```
StringBuffer s1("Hello");
StringBuffer s2("world!");
s1.append(" ");
s1.append(s2);
printf(s1.c_str()); //Hello world!
s1.set("Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
     do eiusmod tempor incididunt ut labore et dolore magna aliqua.");
s1.append("\n");
printf("%zu",s1.size()); //124
s2.clear();
for (int i=0; i<10; i++)
s2.insert(s1,0);
printf(s2.c_str()); //Lorem ipsum ... 10 volte
printf("%zu",s2.size()); //1240
```

## Competenze da acquisire

- Sintassi del linguaggio
- Uso e significato di costruttori e distruttori
- Assegnazione e copia, regola dei tre
- Gestione della memoria
- Incapsulamento