

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

String

```
//=====
class String {
    // VARIABILI DI ISTANZA
    char * base;
    int length;

    //METODI
    // metodi interni di servizio-----
    void strcpy(char * to, const char * from);
    int strlen(const char * s);
    bool strcmp(char * s, char * w);
    void append(char * s, char *w);
```

```
public: //=====
    //METODI
    // metodi essenziali-----
    String ();
    String (char * s);
    String (String & s);
    ~String ();
    String & operator=(const String & s);
    String & operator=(const char * s);
    friend ostream & operator<<(ostream &sx, const String &dx);

    // overloading dell'operatore + -----
    String & operator+(String & right);
    String & operator+(char * s);
    friend String & operator+(char * s,String & t);

    bool String::equals(String & s);
```

String

```

int main() {
    String *s=new String ("pippo");    //A costructor
    String *z=new String();            //Default constructor
    *z="hello";                        //Assignment
    String t("-");                     //A costructor
    //NO! String t="-";                //Copy constructor
    String k;                          //Default constructor
    String w=*s;                       //Copy constructor
    cout<<"*s:"<<*s<<endl;
    cout<<"w:"<<w<<endl;
    cout<<"w==s"<<w.equals(*s)<<endl;
    delete s;
    cout <<"-----"<<endl;
    cout<<"w:"<<w<<endl;
    cout<<"*z:"<<*z<<endl;

```

main

```

    k=*z+w;
    k="<UHU>"+k+t+k+"<UHU>";
    cout<<"k:"<<k<<endl;
    cout<<"*z:"<<*z<<endl;
    *z=k;
    cout<<"*z:"<<*z<<endl;
    delete z;
    cout <<"-----"<<endl;
    cout<<"w:"<<w<<endl;
    k=w+w;
    cout<<"k:"<<k<<endl;
    cout<<"w:"<<w<<endl;
    cout<<"k==w"<<k.equals(w)<<endl;
    cout<<"w==k"<<w.equals(k)<<endl;
}

```

main

```
*s:pippo
```

```
w:pippo
```

```
w==s: 1
```

```
-----
```

```
w:pippo
```

```
*z:hello
```

```
k:<UHU>hellopippo-hellopippo<UHU>
```

```
*z:hello
```

```
*z:<UHU>hellopippo-hellopippo<UHU>
```

```
-----
```

```
w:pippo
```

```
k:pippopippo
```

```
w:pippo
```

```
FAIL:p
```

```
k==w: 0
```

```
w==k: 0
```

OUTPUT

Esercizio 1

Scrivere

il costruttore `String()`,

`String(char * s)`,

il copy constructor e

e il distruttore.

Soluzione 1a

```
String ::String (){
    length=0;
    base=new char[1];
    *base=0;
}
String ::String (char * s){
    // calcola la lunghezza (escluso il carattere di terminazione);
    length=strlen(s);
    // alloca lo spazio
    base=new char[length+1];
    // copia la stringa su base
    strcpy(base,s);
}
```

Soluzione 1b

```
String::String(String & s){ //COPY CONSTRUCTOR
    length=s.length;
    base=new char[length+1];
    strcpy(base,s.base);
}

//-----
String::~String (){ // DISTRUCTOR
    delete []base;
}
```

Esercizio 2

Scrivere

i metodi di servizio

```
void strcpy(char * to, const char * from);  
int strlen(const char * s);  
bool strcmp(char * s, char * w);  
void append(char * s, char *w);
```

Soluzione 2a

```
void String::strcpy(char * to, const char * from) {  
    //Assume che *from sia corretta (null terminated)  
    //Assume che *to abbia allocato uno spazio sufficiente  
    while (*to++ = *from++) ;  
}  
  
int String::strlen(const char * s) {  
    if (s==NULL) return 0;  
    int length=0;  
    while (*(s++)) length++;  
    return length;  
}
```

Soluzione 2b

```
bool String::strcmp(const char * s,const char * w) {  
    //Assume che *s e *w siano corrette (null terminated)  
    do {  
        if (*s++!=*w++) return false;  
    } while (*s);  
    if (*w !=0 ) return false;  
    return true;  
}
```

```
void String::append(char * s, int slen, const char *w) {  
    s+=slen;  
    strcpy(s,w);  
}
```

Esercizio 3

Scrivere

l'operatore di output

e il test di uguaglianza

Semantica:

due String sono uguali se e' uguale il contenuto della stringa (char*) che esse possiedono.

Soluzione 3a: operator<<

```
ostream & operator<<(ostream &sx, const String &s) {  
    sx<<s.base;  
    return sx;  
}
```

Soluzione 3b: comparazione

```
bool String::equals(String &s) {  
    return strcmp(base,s.base);  
}
```

Sub-esercizio:

effettuare

l'overloading di ==

Esercizio 4

Scrivere

l'operatore =

STRING 16

```
String & String::operator=(const String & s){
    if (&s==this) return *this;
    length=s.length;
    delete []base;
    base=new char[length+1];
    strcpy(base,s.base);
    return *this;
}
```

operator=

```
String & String::operator=(const char * s){
    // calcola la lunghezza (escluso il carattere di terminazione);
    length=strlen(s);
    delete []base;
    // alloca lo spazio
    base=new char[length+1];
    // copia la stringa su base
    strcpy(base,s);
    return *this;
}
```

Esercizio 5

Scrivere
le funzioni aggiuntionali

STRING 18

```
bool endsWith(String & s); // confronta la terminazione delle stringhe
bool startsWith(String & s); // confronta l'inizio delle stringhe
int indexOf(char c); //restituisce la posizione della prima occorrenza
                        // del carattere c
int indexOf(char c, int from); //restituisce la posizione della prima
                        // occorrenza del carattere c dopo la posizione from
int indexOf(String s); //restituisce la posizione della prima occorrenza
                        // della sottostringa s
int indexOf(String s, int from); //restituisce la posizione della prima
                        // occorrenza della sottostringa s dopo la posizione from
String subString(int from); //restituisce la sottostringa che inizia
                        // alla posizione from
String subString(int from, int to); //restituisce la sottostringa che va
                        // dalla posizione from alla posizione to
String toLowerCase(); // restituisce una stringa uguale ma in minuscolo
String toUpperCase(); // restituisce una stringa uguale ma in maiuscolo
```

Suggerimento

Per realizzare la funzione toUpperCase, si noti che

$$'a' > 'A'$$
$$'a' = 'A' + ('a' - 'A');$$
$$'b' = 'B' + ('a' - 'A');$$

...

Esercizio 6

Scrivere

l'operatore +

Soluzione 6 a

(Con memory leak!)

STRING 22

```
String & String::operator+(String & right){  
    char * temp=new char[length+right.length+1];  
    char * temp1=temp+length;  
    strcpy(temp,base);  
    strcpy(temp1,right.base);  
    return new String(temp);  
}
```

SBAGLIATA!**operator+**

```
String & String::operator+(String & right){  
    char * temp=new char[length+right.length+1];  
    char * temp1=temp+length;  
    strcpy(temp,base);  
    strcpy(temp1,right.base);  
    String *r=new String(temp);  
    delete temp;  
    return *r;  
}
```

CORRETTA

```
String & String::operator+(char * s){
    String w(s);
    return (*this)+w;
}
```

operator+

```
String & operator+(char * s,String & t){
    String w(s);
    return w+t;
}
```

Nota:
NON e' member function
NON e' friend

```
String & operator+(char * s,char * t){
    String w(s);
    String v(t);
    return w+v;
}
```

NO! **operator+**

[C++ Error] String.cpp(168): E2082 'operator +(char *,char *)' must be a member function or have a parameter of class type.

```
String & String: operator+(char * s,char * t){
    String w(s);
    String v(t);
    return w+v;
}
```

NO!

[C++ Error] String.cpp(178): E2080 'String::operator +(char *,char *)' must be declared with one parameter.

ATTENZIONE!

Espressioni del tipo

String a,b,c;

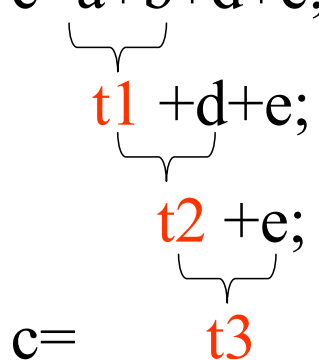
...

c=a+b;

creano oggetti temporanei che non
sono deallocati!

Che succede?

c=a+b+d+e;



t1 +d+e;

t2 +e;

c= t3

Soluzione 6 b

(Senza memory leak)

```
class String {  
    // VARIABILI DI ISTANZA  
    char * base;  
    int length;  
    bool temporary;  
...  
}  
  
String ::String () {  
    ...  
    temporary=false;  
}  
String ::String (const char * s){  
    ...  
    temporary=false;  
}  
  
String::String(String & s){  
    ...  
    temporary=false;  
}
```

```
String & String::operator+(String & right){
    char * temp=new char[length+right.length+1];
    char * temp1=temp+length;
    strcpy(temp,base);
    strcpy(temp1,right.base);
    String *r=new String(temp);
    r->temporary=true;
    delete temp;
    if (this->temporary) delete this;
    return *r;
}
```

Nota:

ultima istruzione prima del return!

```
String & String::operator=(const String & s){
    if (&s==this) return *this;
    length=s.length;
    delete []base;
    base=new char[length+1];
    strcpy(base,s.base);
    if (s.temporary) delete &s;
    return *this;
}
```

Come posso *vedere* che funziona?

Variabili statiche

STRING 32

```
class String {
    // VARIABILI DI ISTANZA
    char * base;   int length;   bool temporary;
    ...
public:
    static int n;
    ...
}
```

**n e' UNA in comune per tutti gli oggetti di tipo String.
E' anche referenziabile dall'esterno come String::n;**

```
int String::n=0;           // va inizializzata come una globale
int main() {
    ...
    cout<<"#Strings: "<<String::n;
}
```

```
String::String () {  
    ...  
    cout<<"C0 : "<<++n;  
}  
String::String (const char * s){  
    ...  
    cout<<"C  : "<<++n;  
}  
  
String::String(String & s){  
    ...  
    cout<<"CC : "<<++n;  
}  
String::~~String(){  
    ...  
    cout<<"D  : "<<- -n;  
}
```