Document Number: X3J16/92-0090-R1
WG21/N0167
Date: November 12, 1992
Project: Programming Language C++
Reply To: Uwe Steinmueller
uwe.steinmueller@ap.mchp.sni.de

A String Class in C++
Revision 1a

Uwe Steinmueller

SNI AP 44

Siemens Nixdorf Informationssysteme AG 1992

1. Major changes from the original paper to Revision 1

- Changed signature for all find functions. They return true on success and false on failure. If found, the position is returned by a size_t& argument.
- New member functions "getRemove" added, which provide a convenient way to get characters from a string, and also remove them. This is quite useful for queue operations on strings.
- If a char* pointer with value 0 is used in an argument, an InvalidArgument exception is thrown.
- If there is an unsigned overflow (adding two size_t values) an LengthError exception is thrown.
- The name "getSubstring" is changed to "substr".

2. Introduction

- This string class is a low level class without any national language support. Language supports is provided by a different class.
- This version assumes value semantics. for strings. (Of course, an implementation may choose to use reference counted pointers, and implement the value semantics with a copy-on-write strategy).
- Conversion to char* (dangling pointers and breaking class boundaries) is dangerous. As we believe there is no realistic way to leave it out (especially when interfacing to C) we prevent the *implicit* casting to char* and support the explicit cast function const char* cStr().
- We decided to permit storage of 0x00 in a string at an arbitrary position. This is sometimes needed (otherwise we would limit the string class unnecessarily).
- Empirical tests have shown that strings which allocate memory in some sort of chunks can perform significantly better in some sort of programs. We do not want to force an implementation to use this stategy, but since a main reason for a standard is to provide better portability, our approach tries to support both ways. An implementation is allowed to ignore all capacity requests. Capacity is just a hint to the implemention it might use or not. In this way an application is portable across different implementations.

- Search using regular expressions should be handled in a different class.
- A class StringStream should be provided.
- Some might argue that the class presented here has far too many features, and many of the extra functions can easily be added by the implementors or the users. But this is exactly the problem: if we know something will be a "common" situation in practice, the standard should guarantee there is an acceptable (what ever this means) solution within the standard.

  We agree that a standard with many member functions is more difficult to understand than a more minimal solution, but we believe that it is far more difficult to understand all the different extensions to the standard.
- We try to use more meaningful names than the ANSI C library.

3. Public interface of the string class

The type size_t is used for the string length. We need an extra value to indicate an invalid position (NPOS). It holds that every valid position is < NPOS. The class Size_T is used as a wrapper to size_t and used in those cases where we want to guarantee that there is a difference for overloading with an integral type and the size_t type for use in the string class. This is important for constructors with just one argument String::String(size_t) versus String::String(Size_T), the first would be used as a implicit conversion from the integral type of size_t to string (which we do not want) and the second is not.

The check for a valid position pos follows the following conventions:

- In all read access operations (normally const member functions) pos has to be < length(). An exception is substr(), it returns an empty string if pos == length().
- In all write access operations (no const member functions) pos has to be <= length(). If pos == length() the write is performed like an append. This allows code like the following:

```
String s;

for(int i = 0; i < 256; i++)
    s.putAt(i, (char)i);
```

- All find functions with pos >= length() return always false.

Names for variables used:

String:	s, s1, s2
Pointer to C string:	cs
Pointer to char* buffer:	cb
size_t:	pos, rep, n, len
size_t&:	fpos
char:	c
ostream:	os
istream:	is
Size_T:	ic

```
class Size_T                    // wrapper for size_t
{
public:
    Size_T(size_t n) : val(n) {}
    size_t value() { return val; }
    ~Size_T() {}
};

//
// public interface of String class
//
```

```
class String
{
    //
    // Exceptions: OutOfMemory, OutOfRange, InvalidArgument
    //             LengthError
    //

public:
    //
    // constructors
    //

    String();
    String(Size_T ic);
    String(const String& s);
    String(const char* cb, size_t n = NPOS);
    String(char c, size_t rep = 1);

    //
    // destructor
    //

    ~String();

    //
    // Assignment (value semantics)
    //

    String& operator=(const String& s);

        // needed for convenience and efficiency

        String& operator=(const char* cs);
        String& assign(char c, size_t rep = 1);
        String& assign(const char* cb, size_t n = NPOS);
        String& operator=(char c);


    //
    // Concatenation
    //

    String& operator+=(const String& s);

        // needed for convenience and efficiency

        String& operator+=(const char* cs);
        String& append(const char* cb, size_t n = NPOS);
        String& append(char c, size_t rep = 1);
        String& operator+=(char c);

    friend String operator+(const String& s1, const String& s2);

        // needed for convenience and efficiency

        friend String operator+(const char* cs, const String& s);
        friend String operator+(const String& s, const char* cs);
        friend String operator+(char c, const String& s);
        friend String operator+(const String& s, char c);
```

```
//
// Comparison / Predicates
//

int compare(const String& s) const;

friend int operator==(const String& s1, const String& s2);
friend int operator!=(const String& s1, const String& s2);

    // needed for convenience and efficiency

    int compare(const char* cb, size_t n = NPOS) const;

    friend int compare(const String& s1, const String& s2);
    friend int operator==(const String& s1, const String& s2);
    friend int operator!=(const String& s1, const String& s2);

    friend int compare(const char* cs, const String& s2);
    friend int operator==(const char* cs, const String& s2);
    friend int operator!=(const char* cs, const String& s2);

    friend int compare(const String& s1, const char* cs);
    friend int operator==(const String& s1, const char* cs);
    friend int operator!=(const String& s1, const char* cs);

//
// Insertion at some pos
//

String& insert(size_t pos, const String& s);

    // needed for convenience and efficiency

    String& insert(size_t pos, const char* cb, size_t n = NPOS);
    String& insert(size_t pos, char c, size_t rep = 1);

//
// Removal
//

String& remove(size_t pos, size_t n = NPOS);

String& getRemove(char& c,size_t pos);
String& getRemove(String &s, size_t pos, size_t n = NPOS);

//
// Replacement at some pos
//

String& replace(size_t pos, size_t n, const String& s);

    // needed for convenience and efficiency

    String& replace(size_t pos, size_t n, const char* cb,
                                    size_t l = NPOS);
    String& replace(size_t pos, size_t n, char c, size_t rep = 1);

//
// Subscripting
//
```

```
char getAt(size_t pos) const;
void putAt(size_t pos, char c);

//
// Search
//

int find(char c, size_t& fpos, size_t pos = 0) const;
int find(const String& s, size_t& fpos, size_t pos = 0) const;
int find(const char* cb, size_t& fpos, size_t pos = 0,
            size_t n = NPOS) const;

int rfind(char c, size_t& fpos, size_t pos = NPOS) const;
int rfind(const String& s, size_t& fpos, size_t pos = NPOS) const;
int rfind(const char* cb, size_t& fpos, size_t pos = NPOS,
            size_t n = NPOS) const;

//
// Substring
//

String substr(size_t pos, size_t n = NPOS) const;

//
// I/O
//

friend ostream& operator<<(ostream& os, const String& s);
friend istream& operator>>(istream& is, String& s);
friend istream& getline(istream& is, String& s, char c = '\n');

// ANSI C functionality

// functionality of strpbrk() and strcspn()

int findFirstOf(const String &s, size_t& fpos, size_t pos = 0) const;
int findFirstOf(const char* cb, size_t& fpos, size_t pos = 0,
                size_t n = NPOS) const;

int findFirstNotOf(const String& s, size_t& fpos, size_t pos = 0)
                                                        const;
int findFirstNotOf(const char* cb, size_t& fpos, size_t pos = 0,
                                        size_t n = NPOS) const;

int findLastOf(const String &s, size_t& fpos,
                                 size_t pos = NPOS) const;
int findLastOf(const char* cb, size_t& fpos, size_t pos = NPOS,
                                size_t n = NPOS) const;

int findLastNotOf(const String& s, size_t& fpos,
                                    size_t pos = NPOS) const;
int findLastNotOf(const char* cb, size_t& fpos, size_t pos = NPOS,
                                    size_t n = NPOS) const;

// an equivalent to strtok is not provided, as this should be
// the task of more powerful special classes

//
// Miscellaneous
//
```

```
    // length

    size_t length() const;

    // copy to C buffer

    size_t copy(char* cb, size_t n, size_t pos = 0);

    // get pointer to internal character array

    const char* cStr() const;

    // Capacity

    size_t reserve() const;
    void   reserve(size_t ic) const;
};
```

4. Description of the public String member functions

All member functions which are only declared for the reason of efficiency or convenience are not decribed here as they do not add any functionality.

4.1. Constructors

Declarations:

`String()`

Synopsis:

Default constructor creates String of length zero.

Pre-conditions:

None

Post-conditions:

`length() == 0`

Result:

None

Exceptions:

OutOfMemory

Declarations:

`String(Size_t)`

Synopsis:

Creates a String of length zero. The implementation may make usage of a capacity value.

Pre-conditions:

None

Post-conditions:

```
length() == 0
```

Result:

None

Exceptions:

OutOfMemory

Declarations:

```
String(const String& s)
```

Synopsis:

Copy constructor creates a String with the value copy of the String s.

Pre-conditions:

None

Post-conditions:

```
length() == s.length()
memcmp(cStr(), s.cStr(), s.length()) == 0
```

Result:

None

Exceptions:

OutOfMemory

Declarations:

```
String(const char *cb, size_t n = NPOS)
```

Synopsis:

If n == NPOS cb is assumed pointing to a null-terminated C-string and a String containing the characters of this C-string is created.

If n < NPOS a String containing the first n elements of the buffer pointed to by cb is created.

If cb is 0 an IvalidArgument exception is thrown

Pre-conditions:

None

Post-conditions:

```
if(n == NPOS)
    length() == strlen(cb)
else
    length() == n
memcmp(cStr(), cb, length()) == 0
```

Result:

None

Exceptions:

OutOfMemory, InvalidArgument

Declarations:

```
String(char c, size_t rep = 1)
```

Synopsis:

Creates a String containing rep times character c.

Pre-conditions:

```
rep < NPOS
```

Post-conditions:

```
length() == rep
for(i = 0; i < rep; i++)
    getAt(i) == c
```

Result:

None

Exceptions:

OutOfMemory, OutOfRange

4.2. Destructor

Declarations:

```
~String();
```

Synopsis:

Destructs the String and frees all unneeded memory.

Pre-conditions:

None

Post-conditions:

None

Result:

None

Exceptions:

None

4.3. Assignment

Declarations:

```
String& operator=(const String& s)
```

Synopsis:

Frees old content (if &s != this) and creates a copy of s. Returns a reference to the target String.

Pre-conditions:

None

Post-conditions:

```
length() == s.length()
memcmp(cStr(), s.cStr(), length()) == 0
```

Result:

Reference to String

Exceptions:

OutOfMemory

4.4. Concatenation

Declarations:

```
String& operator+=(const String& s)
```

Synopsis:

Append content of String to the target String and return a reference to the target.

Pre-conditions:

None

Post-conditions:

```
length() == s.length() + (oldlength = Length(target on entry))
memcmp(cStr() + oldlength, s.cStr(), s.length()) == 0
```

Result:

Reference to String

Exceptions:

OutOfMemory, LengthError

Declarations:

```
friend String operator+(const String& s1, const String& s2)
```

Synopsis:

Concatenate Strings s1 ans s2. Returns a new string with the result.

This function is not needed for its functionality.

There might be in some cases an unacceptable performance overhead due to creation of temporaries. Especially care must be taken in the use of the cStr() member functions.

```
const char *p = (String("/foo") + '/' + "foo.c").cStr();
open(p);          // p is not guaranteed to be valid
```

Returns a String holding the result.

Pre-condition

None

Post-conditions:

```
String s = s1 + s2;
s.length() == (s1.length() + s2.length())
memcmp(s.cStr(), s1.cStr(), s1.length()) == 0
memcmp(s.cStr() + s1.length(), s2.cStr(), s2.length()) == 0
```

Result:

String

Exceptions:

OutOfMemory, LengthError

4.5. Predicates

Declarations:

```
friend int operator== (const String& s1, const String& s2)

anlogous:

friend int operator!= (const String& s1, const String& s2)
```

Synopsis:

Test for equality (not equality) of String s1 with the String s2. Two strings s1 and s2 are assumed to be equal if they have the same length an for all i (0 <= i <= length-1) s1.getAt(i) == s2.getAt(i) holds. Returns a boolean value.

Pre-conditions:

None

Post-conditions:

None

Result:

Bool

Exceptions:

None

4.6. Comparison

Declarations:

```
int compare(const String& s) const
```

Synopsis:

Compares String with the String s. The result should be the same as if the C function memcmp() is performed on the internal representation. It returns an integer less than, equal to, or greater than 0, according as the this string is lexicographically less than, equal to, or greater than s.

Pre-conditions:

```
None
```

Post-conditions:

```
s.compare(s) == 0
```

Result:

int

Exceptions:

InvalidArgument (for the version taking an const char* argument)

4.7. Insert operations

Declarations:

```
String& insert(size_t pos, const String& s)
```

Synopsis:

Insert the String s at Position pos into the target String. If pos > s.length() an OutOfRange exception is thrown. A Reference to the modified target String is returned.

Pre-conditions:

```
pos <= (oldlength = length())
```

Post-conditions:

```
length() == s.length() + oldlength
memcmp(cStr() + pos, s.cStr(), s.length()) == 0
```

Result:

Reference to String

Exceptions:

OutOfMemory, OutOfRange, LengthError, InvalidArgument (for the versions taking an const char* argument)

4.8. Removal

Declarations:

```
String& remove(size_t pos, size_t n = NPOS)
String& getRemove(char& c, size_t pos);
String& getRemove(String& s, size_t pos, size_t n = NPOS);
```

Synopsis:

From the target String len characters starting at position pos are removed. If n == NPOS then len = length() - pos else len = min(n, length() - pos) Of course, with getRemove(char& c, size_t pos) len equals always 1. getRemove assigns the removed character(s) to c, respectively to the String s. A reference to *this is returned.

Pre-conditions:

```
pos != NPOS; pos < (oldlength = length())
```

Post-conditions:

```
length() == oldlength() - len
```

Result:

Reference to String

Exceptions:

OutOfMemory, OutOfRange

4.9. Replace operations

Declarations:

```
String& replace(size_t pos, size_t n, const String& s)
```

Synopsis:

s.replace(pos, n, s) is exactly the same as s.remove(pos, n) followed by s.insert(pos, s) but it can be implemented more efficiently and is more convenient.

Pre-conditions:

```
pos < (oldlength = length())
```

Post-conditions:

```
s1 = s;
s1.remove(pos, n);
s1.insert(pos, s); s1 == s.replace(pos, n, s)
```

Result:

Reference to String

Exceptions:

OutOfMemory, OutOfRange, LengthError

4.10. Subscripting

Declarations:

```
char getAt(size_t pos) const;
void putAt(size_t pos, char c);
```

Synopsis:

If pos is not a valid position an OutOfRange exception is thrown. The member function getAt returns the character at position pos and putAt sets the character at pos to c.

The call of putAt(length(), c) performs like operator+=(c).

Pre-conditions:

```
pos < length()    for getAt
pos <= length()   for putAt
```

Post-conditions:

```
putAt: getAt(pos) == c
```

Result:

getAt (char) and putAt (void)

Exceptions:

OutOfRange

4.11. Find operations

Declarations:

```
int find(char c, size_t& fpos, size_t pos = 0) const
int rfind(char c, size_t& fpos, size_t pos = NPOS) const
```

Synopsis:

All find member functions search for a String, character, or C char* buffer in the target String. If pos is a valid index and the searched for object is found, the return value is true and the value of fpos returns the position where it is found, else false is returned.

If pos is not a valid positon in the string the result is false.

Function rfind searches backwards, NPOS indicates a start at the end of the String.

Pre-conditions:

Post-conditions:

None

Result:

bool

Exceptions:

InvalidArgument (for the versions taking an const char* argument)

4.12. Substring

Declarations:

```
String substr(size_t pos, size_t n = NPOS) const
```

Synopsis:

The getSubstring member function creates a String with the content of len characters in the target String ranging from pos for len characters. If n == NPOS then len = length() - pos else len = min(n, length() - pos).

If pos == length() an empty string is returned.

An OutOfRange exception is thrown if pos > length().

Pre-conditions:

```
pos <= length();
```

Post-conditions:

None

Result:

String

Exceptions:

OutOfRange, OutOfMemory

4.13. String input/output operations

Declarations:

```
friend ostream& operator<<(ostream& os, const String& s)
friend istream& operator>>(istream& is, String& s)
friend istream& getline(istream& is, String& s, char c = '\n')
```

Synopsis:

Operator<< outputs to the ostream os all characters of String s. Also characters containing 0x00 will be written to os.

Operator>> inserts all characters up to the next white space, EOF, or error (without putting any white space to the String s.)

Getline creates a String s containing all character up to the next character c, EOF or error (not containing c itself). The character c is consumed.

Pre-conditions:

`A valid stream`

Post-conditions:

`None`

Result:

ostream& (istream&)

Exceptions:

(see exceptions of the iostream library)

4.14. ANSI C functionality

Declarations:

```
int findFirstOf(const String& s, size_t& fpos, size_t pos = 0) const
int findLastOf(const String& s, size_t& fpos, size_t pos = NPOS) const
```

Synopsis:

Returns true and in fpos the first character which is contained in s or false if not found. If pos is not a valid positon in the string the result is false. The function findLastOf is searching from pos in reverse order moving to position 0.

Pre-conditions:

`None`

Post-conditions:

`None`

Result:

bool

Exceptions:

InvalidArgument (in the versions having an const char* argument)

Declarations:

```
int findFirstNotOf(const String& s, size_t& fpos,
          size_t pos = 0) const
int findLastNotOf(const String& s, size_t& fpos,
          size_t pos = NPOS) const
```

Synopsis:

Returns true and in fpos the first character which is not contained in s or false if not found. If pos is not a valid positon in the string the result is false. The function findLastNotOf is searching from pos in reverse order moving to position 0.

Pre-conditions:

None

Post-conditions:

None

Result:

bool

Exceptions:

InvalidArgument (in the versions having an const char* argument)

4.15. Miscellaneous

Declarations:

```
size_t length() const
```

Synopsis:

Returns the length of the String. As characters 0x00 can be stored in a String length() might be > strlen(cStr()).

Pre-conditions:

None

Post-conditions:

None

Result:

size_t

Exceptions:

None

Declarations:

```
size_t copy(char* cb, size_t n, size_t pos = 0);
```

Synopsis:

There are len = min(n, length() - pos) characters starting at pos are copied to the area pointed to by cb. The client guarantees that the area pointed to by cb holds at minimum n characters. The value of len is returned.

If pos is out of range the OutOfRange exception is thrown.

If cb == 0 the InvalidArgument exception is thrown.

Pre-conditions:

```
pos < length(); //the area pointed to by cb must hold n characters
```

Post-conditions:

```
memcmp(cb, cStr() + pos, len) == 0
```

Result:

size_t

Exceptions:

OutOfRange, InvalidArgument

Declarations:

```
const char* cStr() const
```

Synopsis:

Returns a char* pointer to the internal representation of the String. Nearly all non const member functions may invalidate this pointer. Do not use this function on temporaries. This function guarantees that the string is null-terminated. If the implementation uses a copy-on-write mechanism there should be a member function to get a unique copy of the string. A cast to char* (casting constness away) should not be used, the subscripting functions are to be prefered.

Pre-conditions:

```
The String is not a temporary
```

Post-conditions:

```
Result points to a null-terminated string
```

Result:

char *

Exceptions:

None

Declarations:

```
size_t reserve()
void reserve(Size_T ic)
```

Synopsis:

The reserve() member function returns a value which is determined by the implementation to indicate the current internal storage size. The returned value is always greater or equal then lenght(). The second function gives a hint to the implementation and returns the new capacity. A value ic < length() is ignored.

Pre-conditions:

```
ic != NPOS
```

Post-conditions:

```
return value >= length() and String content unchanged
```

Result:

size_t

Exceptions:

OutOfMemory, OutOufRange

5. References

Plauger, P.J. The Standard C Library. Eddison Wesley 1992.