```
47      else
48       cout << "is not";                                    // Else failure
49      cout << " found in `" << s1 << "\n";                  // Complete output
50      cout << "The pattern begins at zero-relative index " << s1.start ();
51      cout << " and ends at index " << s1.end () << "\n";
52      exit (0);                                             // Exit with OK
53     }
```

Line 1 includes the COOL `Gen_String.h` class header file, and line 2 includes the `Regexp.h` class header file. Lines 4 through 9 perform the assignment and concatenation of a character string to the **Gen_String** object. Lines 10–15 reverse the order of the characters in the object and manipulate the case of the words. Lines 16 through 21 insert, replace, and remove various characters from the object. Lines 22 through 31 demonstrate use of the built-in regular expression function in the **Gen_String** class with the first pattern used in the **Regexp** example program. Lines 32 through 41 demonstrate use of the second pattern, and lines 42 through 51 use the third pattern from the regular expression program. Finally, line 52 ends the program with a successful status.

The following shows the output from the program:

```
s1 reads: Hello
s1 has 5 characters
s1 reads: Hello world!
s1 has 12 characters
s1 backwards reads: !dlrow olleH
s1 upper case: HELLO WORLD!
s1 lower case: hello world!
s1 capitalized: Hello World!
s1 reads: Oh, Hello World!
s1 reads: Oh, Goodbye World!
s1 reads: Oh, World!
The pattern 'Hi There' is found in 'Garbage Hi There garbage'
The patter begins at zero-relative index 8 and ends at index 16
The pattern '[^ab1-9]' is found in 'ab123QQ59ba'
The pattern begins at zero-relative index 5 and ends at index 6
The pattern 'O(.*r)' is found in 'That's OK for me. OK for you?'
The pattern begins at zero-relative index 7 and ends at index 24
```

General String Example

2.9 The following program uses the COOL general string class to show the combined functionality of the previous two classes. A general string object is declared and manipulated with several of the member functions to change its value and size. Several of the overloaded **Gen_String** operators are used to perform concatenation and assignment. After each operation is complete, the resulting string is printed. In addition, several search operations using the built-in regular expression capability are performed. The first is a simple character match, the second a search for a range of characters, and the third a complex match using sub-patterns. Each pattern and string to be searched is printed, along with the ensuing matches and zero-relative index results.

```
1   #include <COOL/Gen_String.h>                  // Include header file
2   #include <COOL/Regexp.h>                      // Include header file

3   int main (void) {
4    Gen_String s1 = "Hello";                                  // Create string
5    cout << "s1 reads: " << s1 << "\n";                       // Display string
6    cout << "s1 has " << strlen (s1) << " characters\n";      // Display count
7    s1 = s1 + " " + "world!";                                 // Join characters
8    cout << "s1 reads: " << s1 << "\n";                       // Display string
9    cout << "s1 has " << strlen (s1) << " characters\n";      // Display count
10   s1.reverse ();                                            // Reverse order
11   cout << "s1 backwards reads: " << s1 << "\n";             // Output string
12   s1.reverse ();                                            // Restore order
13   cout << "s1 upper case: " << upcase (s1) << "\n";         // Uppercase value
14   cout << "s1 lower case: " << downcase (s1) << "\n";       // Downcase value
15   cout << "s1 capitalized: " << capitalize (s1) << "\n";    // Capitalized value
16   s1.insert ("Oh, ", 0);                                    // Insert at start
17   cout << "s1 reads: " << s1 << "\n";                       // Display string
18   s1.replace ("Goodbye", 4, 9);                             // Replace 'hello'
19   cout << "s1 reads: " << s1 << "\n";                       // Display string
20   s1.remove (4, 12);                                        // Remove 'goodbye'
21   cout << "s1 reads: " << s1 << "\n";                       // Display string
22   s1.compile("Hi There");                                   // Define pattern
23   s1 = "Garbage Hi There garbage";                          // Set search string
24   cout << "The pattern 'Hi There' ";                        // Output start
25   if (s1.find () == TRUE)                                   // Pattern found?
26    cout << "is";                                            //  Yes
27   else
28    cout << "is not";                                        // Else failure
29   cout << " found in '" << s1 << "\n";                      // Complete output
30   cout << "The pattern begins at zero-relative index " << s1.start ();
31   cout << " and ends at index " << s1.end () << "\n";
32   s1.compile("[^ab1-9]");                                   // Complex pattern
33   s1 = "ab123QQ59ba";                                       // Search string
34   cout << "The pattern '[^ab1-9]' ";                        // Output start
35   if (s1.find () == TRUE)                                   // Pattern found?
36    cout << "is";                                            //  Yes
37   else
38    cout << "is not";                                        // Else failure
39   cout << " found in '" << s1 << "\n";                      // Complete output
40   cout << "The pattern begins at zero-relative index " << s1.start ();
41   cout << " and ends at index " << s1.end () << "\n";
42   s1.compile("O(.*r)");                                     // New pattern
43   s1 = "That's OK for me. OK for you?";                     // Another string
44   cout << "The pattern 'O(.*r)' ";                          // Output start
45   if (s1.find () == TRUE)                                   // Pattern found?
46    cout << "is";                                            //  Yes
```

friend Gen_String& strcpy (**Gen_String&** *str1*, **const Gen_String&** *str2*);
Copies one general string object *str2* into another general string object *str1*. This function returns a reference to the modified general string object *str1*.

inline friend long strlen (**const Gen_String&** *str*);
Returns the number of characters (length) of the general string *str*.

friend Gen_String& strncat (**Gen_String&** *str1*, **const char*** *str2*, **int** *n*);
Concatenates some number of characters *n* from a character string *str2* to a general string object *str1*. This function returns a reference to the modified general string object *str1*. If a negative length is specified, an **Error** exception is raised.

friend Gen_String& strncat (**Gen_String&** *str1*, **const Gen_String&** *str2*, *int n*);
Concatenates some number of characters *n* from one general string object *str2* to another general string object *str1*. This function returns a reference to the modified general string object *str1*. If a negative length is specified, an **Error** exception is raised.

friend Gen_String& strncpy (**Gen_String&** *str1*, **const char*** *str2*, *long n*);
Copies some number of characters *n* from the character string *str2* into the general string object *str1*. This function returns a reference to the modified general string object *str1*. If a negative number is specified, an **Error** exception is raised.

friend char* strrchr (**const Gen_String&** *str,* **char** *c*);
Overloads the backward character-search function to scan from right to left through a general string object *str* for the last occurrence of a specific character *c*. This function returns a pointer to the character if found; otherwise, this function returns **NULL**.

friend double strtod (**const Gen_String&** *str*, **char**** *ptr* = **NULL**);
Returns the double floating-point value represented by the characters in the general string object *str*. If the second argument is non-zero, it is set to the character terminating the converted string value.

friend long strtol (**const Gen_String&** *str*, **char**** *ptr*=**NULL**, *int radix*=10);
Returns the long number represented by the characters in the general string object *str*. If no specific radix is specified, the default radix is decimal. If the second argument is non-zero, it is set to the character terminating the converted string value.

friend Gen_String& trim (**Gen_String&** *str1*, **const char*** *str2*);
Removes any occurrence of the character string *str2* in the general string object *str1*. This function returns a reference to the modified general string *str1*.

friend Gen_String& upcase (**Gen_String&** *str*);
Converts any alphabetic character to uppercase. This function returns a reference to the modified general string *str*.

friend int atoi (**const Gen_String&** *str*);
Returns the decimal radix integer number represented by the characters in the general string object *str*.

friend long atol (**const Gen_String&** *str*);
Returns the decimal radix long number represented by the characters in the general string object *str*.

friend Gen_String& capitalize (**Gen_String&** *str*);
Capitalizes each word and returns the modified general string *str*. A word is defined to be any subsequence of alphanumeric characters, but it must start with a letter. This function returns a reference to the modified general string.

friend Gen_String& downcase (**Gen_String&** *str*);
Converts any alphabetic character to lowercase. This function returns a reference to the modified general string *str*.

friend Gen_String& left_trim (**Gen_String&** *str1*, **const char*** *str2*);
Removes any prefix occurrence of the character string *str2* in the general string object *str1*. This function returns a reference to the modified general string *str1*.

friend ostream& operator<< (**ostream&** *os*, **const Gen_String&** *str*);
Overloads the output operator for a reference to a general string object.

inline friend ostream& operator<< (**ostream&** *os*, **const Gen_String*** *str*);
Overloads the output operator for a pointer to a general string object.

friend Gen_String& right_trim (**Gen_String&** *str1*, **const char*** *str2*);
Removes any suffix occurrence of the character string *str2* in the general string object *str1*. This function returns a reference to the modified general string *str1*.

friend Gen_String& strcat (**Gen_String&** *str*, **char** *c*);
Concatenates a single character *c* to a general string object *str*. This function returns a reference to the modified general string *str*.

friend Gen_String& strcat (**Gen_String&** *str1*, **const char*** *str2*);
Concatenates a copy of the character string *str2* to a general string object *str1*. This function returns a reference to the modified general string *str1*.

friend Gen_String& strcat (**Gen_String&** *str1,* **const Gen_String&** *str2*);
Concatenates one general string object *str2* to another general string object *str1*. This function returns a reference to the modified general string *str1*.

friend char* strchr (**const Gen_String&** *str,* **char** *c*);
Overloads the forward character-search function to scan from left to right through a general string object *str* for the first occurrence of a specific character *c*. This function returns a pointer to the character if found; otherwise, this function returns **NULL**.

friend Gen_String& strcpy (**Gen_String&** *str*, **char** *c*);
Returns the result of copying a single character into a general string object *str*. This function returns a reference to the modified general string object *str*.

friend Gen_String& strcpy (**Gen_String&** *str1***, const char*** *str2*);
Copies a character string *str2* into a general string object *str1*. This function returns a reference to the modified general string object *str1*.

inline operator char* () **const**;
Provides an implicit conversion operator to convert a string object into a **char*** value.

Boolean remove (**long** *start*, **long** *end*);
Removes the sequence of characters between the zero-relative inclusive *start* and exclusive *end* indexes provided. This function returns **TRUE** if successful; otherwise, this function returns **FALSE** if either one or both of the indexes is out of range.

Boolean replace (**const char*** *str*, **long** *start*, **long** *end*);
Replaces the sequence of characters between the zero-relative inclusive *start* and exclusive *end* indexes with a copy of the character string *str*. This function returns **TRUE** if successful; otherwise, this function returns **FALSE** if either one or both of the indexes is out of range.

void resize (**long** *size*);
Resizes the general string object to hold at least *size* characters. If a negative size is specified, an **Error** exception is raised.

void reverse ();
Reverses the ordering of the characters in a general string object. The **Reg_Exp** does not need to be recompiled.

inline void set_alloc_size (**int** *size*);
Updates the allocation growth size to be used when the growth ratio is zero. Default allocation growth size is 100 bytes. If a negative size is specified, an **Error** exception is raised.

inline void set_growth_ratio (**float** *ratio*);
Updates the growth ratio for this instance of a **Gen_String** class object to the specified value. When a string needs to grow, the current size is multiplied by the ratio to determine the new size. If a negative growth ratio is specified, an **Error** exception is raised.

inline long start () **const**;
Returns an index into the last string successfully searched for by that object. The index corresponds to the beginning of the last item found.

void sub_string (**Gen_String&** *str*, **long** *start*, **long** *end*);
Sets the general string object *str* to the values of the character sequence between the zero-relative inclusive *start* and exclusive *end* indexes provided. This function returns **TRUE** if successful; otherwise, if the start or end indexes or both are out of range, an **Error** exception is raised and, if the handler returns, this function returns **FALSE**.

void yank (**Gen_String&** *str*, **long** *start*, **long** *end*);
Deletes the sequence of characters between the zero-relative inclusive *start* and exclusive *end* indexes provided and sets the string object *str* to the value of the deleted characters. If the start or end or both indexes are out of range, an **Error** exception is raised.

Friend Functions:

inline friend double atof (**const Gen_String&** *str);*
Returns the floating-point value represented by the characters in the general string object *str*.

inline Boolean operator== (**const Gen_String&** *str*) **const**;
Overloads the equality operator for the **Gen_String** class. This function returns **TRUE** if the general strings have the same sequence of characters; otherwise, this function returns **FALSE**.

inline Boolean operator!= (**const char*** *str*) **const**;
Overloads the inequality operator for the **Gen_String** class. This function returns **FALSE** if the general string object and *str* have the same sequence of characters; otherwise, this function returns **TRUE**.

inline Boolean operator!= (**const Gen_String&** *str*) **const**;
Overloads the inequality operator for the **Gen_String** class. This function returns **FALSE** if the general strings have the same sequence of characters; otherwise, this function returns **TRUE**.

inline Boolean operator< (**const char*** *str*) **const**;
Overloads the less-than operator for the **Gen_String** class. This function returns **TRUE** if the general string object is lexically less than *str*; otherwise, this function returns **FALSE**.

inline Boolean operator< (**const Gen_String&** *str*) **const**;
Overloads the less-than operator for the **Gen_String** class. This function returns **TRUE** if the general string object is lexically less than *str*; otherwise, this function returns **FALSE**.

inline Boolean operator<= (**const** *c***har*** *str*) **const**;
Overloads the less-than-or-equal operator for the **Gen_String** class. This function returns **TRUE** if the general string object is lexically less than or equal to *str*; otherwise, this function returns **FALSE**.

inline Boolean operator<= (**const Gen_String&** *str*) **const**;
Overloads the less-than-or-equal operator for the **Gen_String** class. This function returns **TRUE** if the general string object is lexically less than or equal to *str*; otherwise, this function returns **FALSE**.

inline Boolean operator> (**const char*** *str*) **const**;
Overloads the greater-than operator for the **Gen_String** class. This function returns **TRUE** if the general string object is lexically greater than *str*; otherwise, this function returns **FALSE**.

inline Boolean operator> (**const Gen_String&** *str*) **const**;
Overloads the greater-than operator for the **Gen_String** class. This function returns **TRUE** if the general string object is lexically greater than *str*; otherwise, this function returns **FALSE**.

inline Boolean operator>= (**const char*** *str*) **const**;
Overloads the greater-than-or-equal operator for the **Gen_String** class. This function returns **TRUE** if the general string object is lexically greater than or equal to *str*; otherwise, this function returns **FALSE**.

inline Boolean operator>= (**const Gen_String&** *str*) **const**;
Overloads the greater-than-or-equal operator for the **Gen_String** class. This function returns **TRUE** if the general string object is lexically greater than or equal to *str*; otherwise, this function returns **FALSE**.

inline char operator[] (**long** *position*) **const**;
Returns the character at the zero-relative index *position* into the general string. If an invalid index is specified, an **Error** exception is raised.

Boolean insert (**const char*** *str*, **long** *position*);
Inserts a copy of the sequence of characters *str* at the zero-relative index *position*. This function returns **TRUE** if successful; otherwise, this function returns **FALSE** if the index *position* is out of range.

inline Boolean is_valid () **const**;
Returns **TRUE** if a valid regular expression is compiled and ready for use; otherwise, this function returns **FALSE**.

Gen_String operator+ (**char** *c*);
Overloads the addition operator to concatenate a single character *c* to a general string object. This function returns a new general string object.

Gen_String operator+ (**const char*** *str*);
Overloads the addition operator to concatenate a copy of the character sequence *str* to a general string object. This function returns a new general string object.

Gen_String operator+ (**const Gen_String&** *str*);
Overloads the addition operator to concatenate the value of another general string object *str* to a general string object. This function returns a new general string object.

inline Gen_String& operator= (**char** *c);*
Overloads the assignment operator to assign a single character *c* to a general string object. This function returns a reference to the modified general string object.

inline Gen_String& operator= (**const char*** *str*);
Overloads the assignment operator to assign a copy of the character sequence *str* to a general string object. This function returns a reference to the modified general string object.

inline Gen_String& operator= (**const Gen_String&** *str*);
Overloads the assignment operator to assign the value of another general string object *str* to a general string object. This function returns a reference to the modified general string object.

Gen_String& operator+= (**char** *c*);
Overloads the addition-and-assignment operator to concatenate a single character *c* to a general string object. This function returns a reference to the modified general string object.

Gen_String& operator+= (**const char*** *str*);
Overloads the addition-and-assignment operator to concatenate a copy of the character sequence *str* to a general string object. This function returns a reference to the modified general string object.

Gen_String& operator+= (**const Gen_String&** *str*);
Overloads the addition-and-assignment operator to concatenate the value of another general string object *str* to a general string object. This function returns a reference to the modified general string object.

inline Boolean operator== (**const char*** *str*) **const**;
Overloads the equality operator for the **Gen_String** class. This function returns **TRUE** if the general string object and *str* have the same sequence of characters; otherwise, this function returns **FALSE**.

Name: **Gen_String** — Dynamic general-purpose strings with reference counting, delayed copy, and regular expression pattern matching

Synopsis: **#include** <COOL/Gen_String.h>

Base Classes: **Generic**

Friend Classes: None

Constructors:

Gen_String ();
Initializes an empty general string object with the default size block of memory allocated to hold 100 characters.

Gen_String (**char** *c*);
Initializes a general string object with the default size block of memory allocated to hold 100 characters whose value is the general string consisting of the single character *c*.

Gen_String (**const char*** *str*);
Initializes a general string object with a default size block of memory allocated to hold 100 characters whose value is a copy of the specified character string argument *str*. If *str* is longer than 100 char then the **Gen_String** will grow to the correct size.

Gen_String *(***const char*** *str*, **long** *size*);
Allocates an initial block of memory of size *size* or size **strlen**(*str)* whichever is longer. Initializes the general string object with a copy of the specified character string *str*.

Gen_String (**const Gen_String&** *str*);
Duplicates the size and value of another general string object *str*.

Gen_String (**const Gen_String&** *str*, **long** *size*);
Duplicates the size and value of another general string object *str* and allocates an initial block of memory of size *size* or size **strlen**(*str*) whichever is longer.

Member Functions:

inline long capacity () **const**;
Returns the maximum number of characters that the general string object can contain without having to grow.

void clear ();
Resets the **NULL** character sting terminator to the beginning of the general string and sets the length of the general string to zero.

void compile (**char*** *str*);
Creates a compiled version of the regular expression argument *str*. If an invalid expression is detected, an Error exception is raised. Note that you must recompile a regular expression after changing the value of the **Gen_String** object.

inline long end () **const**;
Returns an index into the last string successfully searched for by this object. The index corresponds to the character after the last item found, or if none was found (the uninitialized state) then its value is **NULL**.

Boolean find ();
Searches for the first or next established regular expression in the string. If the expression is found, this function returns **TRUE** and sets start and end appropriately. If an invalid expression is detected, an Error exception is raised.

```
16      if (r1.find (dummy) == TRUE)                    // Pattern found in string?
17       cout << "is";                                   // Yes, indicate affirmative
18      else
19       cout << "is not";                               // Else indicate failure
20      cout << " found in `" << dummy << "\n";         // And complete output
21      cout << "The pattern begins at zero-relative index " << r1.start ();
22      cout << " and ends at index " << r1.end () << "\n";
23      r1.compile("O(.*r)");                           // New complex pattern
24      strcpy (dummy,"That's OK for me. OK for you?");// Another string to search
25      cout << "The pattern `O(.*r)' ";                // Output start of sentence
26      if (r1.find (dummy) == TRUE)                    // Pattern found in string?
27       cout << "is";                                   // Yes, indicate affirmative
28      else
29       cout << "is not";                               // Else indicate failure
30      cout << " found in `" << dummy << "\n";         // And complete output
31      cout << "The pattern begins at zero-relative index " << r1.start ();
32      cout << " and ends at index " << r1.end () << "\n";
33      exit (0);                                       // Exit with OK status
34     }
```

Line 1 includes the COOL `Regexp.h` class header file. Lines 3 and 4 define a simple regular expression object and a character string pattern to be searched. Lines 5 through 12 search the character string for the pattern, output the search results, and indicate the zero-relative start and end points. Line 13 sets a more complex pattern for the regular expression object. This says to match anything that does not begin with the characters `"ab"` followed by numbers in the series one through nine. Lines 15 through 22 search the character string for this pattern, output the search results, and indicate the zero-relative start and end points. Line 23 establishes a pattern that matches a character string beginning with `"O"`, followed by a sequence of zero or more characters that ends with `"r"`. Lines 25 through 32 search the character string for this pattern, output the search results, and indicate the zero-relative start and end points.

The following shows the output from the program:

```
The pattern 'Hi There' is found in 'Garbage Hi There garbage'
The pattern begins at zero-relative index 8 and ends at index 16
The pattern '[^ab1-9]' is found in 'ab123QQ59ba'
The pattern begins at zero-relative index 5 and ends at index 6
The pattern 'O(.*r)' is found in 'That's OK for me. OK for you?'
The pattern begins at zero-relative index 7 and ends at index 24
```

General String Class

2.8 The **Gen_String** class provides general-purpose, dynamic strings for a C++ application with support for reference counting, delayed copy, and regular expression pattern matching. The intent is to provide sophisticated character string functionality for the application programmer. As in the **String** class, interface and member functions provide typical string operations. These include concatenation, case-sensitive and case-insensitive lexical comparison, string search, yank, delete, and replacement. In addition, the inclusion of regular expression pattern matching facilitates easier use of this COOL class with character strings. The **Gen_String** class is dynamic in the sense that if an operation such as concatenate results in more characters than can fit in the currently allocated memory, the string object *grows* according to some established size or ratio value. System-provided functions for **char*** such as **strcpy** and **strcmp** are also available via the overloaded **operator char*** member function.

inline long start () **const**;

Returns an index into the last character string successfully searched for by this object. The index corresponds to the beginning of the last item found. If none are found, its value is **NULL**.

Regular Expression Example

2.7 The following program utilizes the COOL regular expression class to set and search for several patterns. The first is a simple character match, the second a search for a range of characters, and the third a complex match using sub-patterns. Each pattern and string to be searched is printed, along with the ensuing matches and zero-relative index results.

```
1   #include <COOL/Regexp.h>                    // Include Regexp header file

2   int main (void) {
3    Regexp r1("Hi There");                     // Define simple pattern
4    char* dummy = "Garbage Hi There garbage";  // Dummy string to search
5    cout << "The pattern 'Hi There' ";         // Output start of sentence
6    if (r1.find (dummy) == TRUE)               // Pattern found in string?
7     cout << "is";                            // Yes, indicate affirmative
8    else
9     cout << "is not";                        // Else indicate failure
10   cout << " found in '" << dummy << "\n";    // And complete output
11   cout << "The pattern begins at zero-relative index " << r1.start ();
12   cout << " and ends at index " << r1.end () << "\n";
13   r1.compile("[^ab1-9]");                    // Complex pattern
14   strcpy (dummy, "ab123QQ59ba");             // Another string to search
15   cout << "The pattern '[^ab1-9]' ";         // Output start of sentence
```

Name: **Regexp** — Regular expression pattern matching

Synopsis: **#include** <COOL/Regexp.h>

Base Classes: **Generic**

Friend Classes: None

Constructors:

inline Regexp ();
Creates an empty regular expression object with no private data initialized.

inline Regexp (**char*** *str*);
Creates a regular expression object and initializes all the data by compiling the regular expression provided *str*. If an invalid regular expression is detected, an Error exception is raised.

Regexp (**const Regexp&** *reg*);
Creates a regular expression object and duplicates the values and regular expression of another regular expression object *reg*.

Member Functions:

void compile (**char*** *str*);
Creates a compiled version of the argument *str* and places it in the private data. If an invalid expression is detected, an Error exception is raised.

Boolean deep_equal (**const Regexp&** *reg*) **const**;
Determines if two regular expressions are the same, including the zero-relative start and end indexes of the last successful pattern match. This function returns **TRUE** if the expressions are the same; otherwise, this function returns **FALSE**.

inline long end () **const**;
Returns an index into the last character string successfully searched for by this object. The index corresponds to the character after the last item found. If none are found, its value is **NULL**.

Boolean find (**char*** *str*);
Searches for the already specified regular expression in *str*. If the expression is found, this function returns **TRUE** and sets start and end indexes appropriately. If an invalid expression is detected, an Error exception is raised.

inline Boolean is_valid () **const**;
Returns **TRUE** if a valid regular expression is compiled and ready for use; otherwise, this function returns **FALSE**.

Boolean operator== (**const Regexp&** *reg*) **const**;
Determines if two regular expression objects are the same. This function returns **TRUE** if the expression objects are the same; otherwise, this function returns **FALSE**.

inline Boolean operator!= (**const Regexp&** *reg*) **const**;
Determines if two regular expression objects are not the same. This function returns **TRUE** if the expression objects are different; otherwise, this function returns **FALSE**.

inline void set_invalid ();
Invalidates the current regular expression of the regular expression object.

char* strndup (const char* *str,* **long** *position*);**
Duplicates into a new character string allocated off the heap the sequence of characters from the beginning of *str* to the zero-relative index *position*. This function returns a pointer to the duplicated character string if successful; otherwise, this function returns **NULL** if the index is out of range.

char* strnremove (char* *str,* **long** *position*);**
Removes the sequence of characters from the beginning of *str* to the zero-relative index *position*. This function returns a pointer to the new character string if successful; otherwise, this function returns **NULL** if the index is out of range.

char* stryank (char* *str,* **long** *position*);**
Deletes the sequence of characters from the beginning of *str* to the zero-relative index *position* and allocates a new character string off the heap whose value is the deleted characters. This function returns a pointer to the yanked string if successful; otherwise, this function returns **NULL** if the index is out of range.

void reverse (char* *str*);**
Reverses the order of characters in a string *str*.

Regular Expression (Regexp) Class

2.6 A regular expression allows a programmer to specify complex patterns that can be searched for and matched against the character string of a string object. In its simplest form, a regular expression is a sequence of characters used to search for exact character matches. However, many times the exact sequence to be found is not known, or only a match at the beginning or end of a string is desired. The COOL regular expression class implements regular expression pattern matching as is found and implemented in many UNIX commands and utilities.

The regular expression class provides a convenient mechanism for specifying and manipulating regular expressions. The regular expression object allows specification of such patterns by using the following regular expression meta-characters:

^	Matches at beginning of a line
$	Matches at end of a line
.	Matches any single character
[ ]	Matches any character(s) inside the brackets
[^]	Matches any character(s) *not* inside the brackets
–	Matches any character in range on either side of a dash
*	Matches preceding pattern zero or more times
+	Matches preceding pattern one or more times
?	Matches preceding pattern zero or once only
()	Saves a matched expression and uses it in a later match

Note that more than one of these metacharacters can be used in a single regular expression in order to create complex search patterns. For example, the pattern `[^ab1-9]` says to match any character sequence that does not begin with the characters `"ab"` followed by numbers in the series one through nine.

Boolean is_le (const char* *str1***, const char*** *str1***, Boolean** *case_flag***);**
This function returns **TRUE** if *str1* is lexically less than or equal to *str2*; otherwise, this function returns **FALSE**. If *case_flag* is **TRUE**, a case-sensitive comparison is made; otherwise, a case-insensitive comparison is made.

Boolean is_lt (const char* *str1***, const char*** *str2***, Boolean** *case_flag***);**
This function returns **TRUE** if *str1* is lexically less than *str2*; otherwise, this function returns **FALSE**. If the *Boolean* value is **TRUE**, a case-sensitive comparison is made; otherwise, a case-insensitive comparison is made.

char* c_capitalize (char* *str***);**
Capitalizes each word and returns a pointer to the modified character string *str*. A word is defined to be any subsequence of alphanumeric characters.

char* c_downcase (char* *str***);**
Converts any alphabetic character to lowercase. This function returns a pointer to the modified character string *str*.

char* c_left_trim (char* *str1***, const char*** *str2***);**
Removes any prefix occurrence of the second character string *str2* in the first character string *str1*. This function returns a pointer to the modified character string *str1*.

char* c_right_trim (char* *str1***, const char*** *str2***);**
Removes any suffix occurrence of the second character string *str2* in the first character string *str1*. This function returns a pointer to the modified character string *str1*.

char* c_trim (char* *str1***, const char*** *str2***);**
Removes any occurrence of the second character string *str2* from the first character string *str1*. This function returns a pointer to the modified character string *str1*.

char* c_upcase (char* *str***);**
Converts any alphabetic character to uppercase. This function returns a pointer to the modified character string *str*.

char* strfind (const char* *str1,* **const char*** *str2,* **long&** *start* = 0*,*
long& *end* = 0**);**
This function provides simple pattern matching by scanning from left to right through *str1* for the first occurrence of *str2*. An asterisk can be used to match for zero or more characters and a question mark can be used to match for any single character. This function returns a pointer to the start of the matching character string and updates the zero-relative *start* and *end* indexes if found; otherwise, this function returns **NULL**.

char* strrfind (const char* *str1,* **const char*** *str2,* **long&** *start* = 0*,*
long& *end* = 0**);**
This function provides simple pattern matching by scanning from right to left through *str1* for the first occurrence of *str2*. An asterisk can be used to match for zero or more characters and a question mark can be used to match for any single character. This function returns a pointer to the start of the matching character string and updates the zero-relative *start* and *end* indexes if found; otherwise, this function returns **NULL**.

The following shows the output from the program:

```
s1 reads: Hello
s1 has 5 characters
s1 reads: Hello world!
s1 has 12 characters
s1 backwards reads: !dlrow olleH
s1 upper case: HELLO WORLD!
s1 lower case: hello world!
s1 capitalized: Hello World!
s1 reads: Oh, Hello World!
s1 reads: Oh, Goodbye World!
s1 reads: Oh, World!
```

Auxiliary char* Functions

2.5 The ANSI C specification requires a collection of standard functions for the manipulation of character strings. However, these do not include some of the more useful functions found in the **String** and **Gen_String** classes. In addition, many generic character string functions can be used for the **String** and **Gen_String** objects because of the implicit **operator char***. For these reasons, the following auxiliary functions are defined for the built-in **char*** data type:

Name: **char*** — Auxiliary character string functionality

Synopsis: **#include** <COOL/char.h>

Friend Functions:

Boolean is_equal (const char* *str1***, const char*** *str2***, Boolean** *case_flag* **= FALSE);**

Compares two character strings for lexical equality. If *case_flag* is **TRUE**, a case-sensitive comparison is made; otherwise, a case-insensitive comparison is made. This function returns **TRUE** if the strings are lexically equivalent; otherwise, this function returns **FALSE**.

Boolean is_not_equal (const char* *str1***, const char*** *str1***, Boolean** *case_flag* **= FALSE);**

Compares two character strings for lexical inequality. If *case_flag* is **TRUE**, a case-sensitive comparison is made; otherwise, a case-insensitive comparison is made. This function returns **FALSE** if the strings are lexically equivalent; otherwise, this function returns **TRUE**.

Boolean is_equal_n (const char* *str1***, const char*** *str1***, int** *n***, Boolean** *case_flag* **= FALSE);**

Compares *n* characters in two character strings for lexical equality. If *case_flag* is **TRUE**, a case-sensitive comparison is made; otherwise, a case-insensitive comparison is made. This function returns **TRUE** if the strings are lexically equivalent; otherwise, this function returns **FALSE**.

Boolean is_ge (const char* *str1***, const char*** *str2***, Boolean** *case_flag***);**

This function returns **TRUE** if *str1* is lexically greater than or equal to *str2*; otherwise, this function returns **FALSE**. If *case_flag* is **TRUE**, a case-sensitive comparison is made; otherwise, a case-insensitive comparison is made.

Boolean is_gt (const char* *str1***, const char*** *str1***, Boolean** *case_flag***);**

This function returns **TRUE** if *str1* is lexically greater than *str2*; otherwise, this function returns **FALSE**. If *case_flag* is **TRUE**, a case-sensitive comparison is made; otherwise, a case-insensitive comparison is made.

friend long strtol (**const String&** *str*, **char**** *ptr* = **NULL**, **int** *radix*=10);
Returns the long number represented by the characters in the string object *str*. If a specific radix is not specified, the default radix is decimal. If the second argument is non-zero, it is set to the character terminating the converted string value.

friend String& trim (**String&** *str1*, **const char*** *str2*);
Removes any occurrence of the character string *str2* from the string object *str1*. This function returns a reference to the modified string *str1*.

friend String& upcase (*S***tring&** *str*);
Converts any alphabetic character to uppercase. This function returns a reference to the modified string *str*.

String Example

2.4 The following program declares a string object and manipulates it with several member functions to change its value and size. Several of the overloaded **String** operators are used to perform concatenation and assignment. After each operation is complete, the resulting string is printed.

```
1   #include <COOL/String.h>

2   int main (void) {
3     String s1 = "Hello";                                    // Create string
4     cout << "s1 reads: " << s1 << "\n";                     // Display string
5     cout << "s1 has " << strlen (s1) << " characters\n";    // Display count
6     s1 = s1 + " " + "world!";                              // Concatenate
7     cout << "s1 reads: " << s1 << "\n";                     // Display string
8     cout << "s1 has " << strlen (s1) << " characters\n";    // Display count
9     s1.reverse ();                                         // Reverse order
10    cout << "s1 backwards reads: " << s1 << "\n";           // Output string
11    s1.reverse ();                                         // Restore order
12    cout << "s1 upper case: " << upcase (s1) << "\n";       // Uppercase value
13    cout << "s1 lower case: " << downcase (s1) << "\n";     // Downcase value
14    cout << "s1 capitalized: " << capitalize (s1) << "\n";  // Capitalized value
15    s1.insert ("Oh, ", 0);                                 // Insert at start
16    cout << "s1 reads: " << s1 << "\n";                     // Display string
17    s1.replace ("Goodbye", 4, 9);                          // Replace 'hello'
18    cout << "s1 reads: " << s1 << "\n";                     // Display string
19    s1.remove (4, 12);                                     // Remove 'goodbye'
20    cout << "s1 reads: " << s1 << "\n";                     // Display string
21    exit (0);                                              // Exit with OK
22    }
```

Line 1 includes the COOL `String.h` class header file. Line 3 declares a new string object and initializes it with the word `Hello`. Lines 4 and 5 output the value of the string and the number of characters it contains. Line 6 uses the overloaded **operator+** to concatenate a space and the word `World` to the string object. The result and new character count is then output in lines 7 and 8. Line 9 uses the **reverse()** member function to reverse the order of the letters in the string object. Again, the results are output in line 10, and then reverted back in line 11 with another call to the same reverse member function. Lines 12 through 14 change the case and output the value of the string object. Line 15 adds some letters to the string object by using the **insert()** member function, and line 16 outputs the result. Lines 17 through 19 replace and then remove letters from the string object with the result outputted after each operation. Finally, the program ends with a valid exit code.

friend String& strcat (**String&** *str*, **char** *c*);
Returns the result of concatenating the character *c* to a string object *str*.

friend String& strcat (**String&** *str1*, **const char*** *str2*);
Returns the result of concatenating a copy of the specified character string *str2* to a string object *str1*.

friend String& strcat (**String&** *str1*, **const String&** *str2*);
Returns the result of concatenating one string object *str2* to another *str1*. This function returns the modified string object *str1*.

friend char* strchr (**const String&** *str,* **char** *c*);
Overloads the forward character search function to scan from left to right through a string object *str* for the first occurrence of the character *c*. This function returns a pointer to the character if found; otherwise, this function returns **NULL**.

friend String& strcpy (**String&** *str1*, **char** *str2*);
Overloads the copy string function to copy the character string *str2* into a string argument *str1*. This function returns a reference to the modified string object *str1*.

friend String& strcpy (**String&** *str1*, **const char*** *str2*);
Overloads the copy string function to copy the specified character string *str2* argument into the string argument *str1*. This function returns a reference to the modified string object *str1*.

friend String& strcpy (**String&** *str1*, **const String&** *str2*);
Overloads the copy string function to copy the second string argument *str2* into the first string argument *str1*. This function returns the modified string object *str1*.

inline friend long strlen (**const String&** *str*);
Returns the number of characters (length) of the string *str*.

friend String& strncat (**String&** *str1*, **const char*** *str2*, **int** *length*);
Returns the result of concatenating a copy of some number of characters *length* from a character string *str2* to a string object *str1*. This function returns a reference to the modified string object *str1*.

friend String& strncat (**String&** *str1*, **const String&** *str2*, **int** *length*);
Returns the result of concatenating some number of characters *length* from one string object *str2* to another *str1*. This function returns a reference to the modified string object *str1*.

friend String& strncpy (**String&** *str1*, **const char*** *str2*, **long** *length*);
Overloads the **strncpy** function to copy some number of characters *length* from the specified character string argument *str2* into the string argument *str1*. This function returns a reference to the modified string object *str1*.

friend char* strrchr (**const String&** *str,* **char** *c*);
Overloads the backward character search function to scan from right to left through a string object *str* for the last occurrence of a specific character *c*. This function returns a pointer to the character if found; otherwise, this function returns **NULL**.

friend double strtod (**const String&** *str*, **char**** *ptr* = **NULL**);
Returns the double floating-point value represented by the characters in the string object *str*. If the second argument is non-zero, it is set to the character terminating the converted string value.

void reverse ();
Reverses the ordering of the characters in a string object.

inline void set_alloc_size (**int** *size*);
Updates the allocation growth size to be used when the growth ratio is zero. Default allocation growth size is 100 bytes. If *size* is negative, an Error exception is raised.

inline void set_growth_ratio (**float** *ratio*);
Updates the growth ratio for this instance of a string to the specified value. When a string needs to grow, the current size is multiplied by the ratio to determine the new size. If *ratio* is negative, an **Error** exception is raised.

void sub_string (**String&** *str*, **long** *start*, **long** *end*);
Sets the given string object *str* to the values in the character sequence between the zero-relative inclusive *start* and exclusive *end* indexes provided. This function returns **TRUE** if successful; otherwise, this function returns **FALSE** if either one or both of the indexes is out of range.

void yank (**String&** *str*, **long** *start*, **long** *end*);
Deletes the sequence of characters between the zero-relative inclusive *start* and exclusive *end* indexes provided and sets the given string object *str* to the value of the deleted characters.

Friend Functions:

inline friend double atof (**const String&** *str*);
Returns the floating-point value represented by the characters in the string object *str*.

friend int atoi (**const String&** *str*);
Returns the decimal radix integer number represented by the characters in the string object *str*.

friend long atol (**const String&** *str*);
Returns the decimal radix long number represented by the characters in the string object *str*.

friend String& capitalize (**String&** *str*);
Capitalizes each word and returns the modified string *str*. A word is defined to be any subsequence of alphanumeric characters. This function returns a reference to the modified string *str*.

friend String& downcase (**String&** *str*);
Converts any alphabetic character to lowercase. This function returns a reference to the modified string *str*.

friend String& left_trim (**String&** *str1*, **const char*** *str2*);
Removes any prefix occurrence of the character string *str2* specified from the string object *str1*. This function returns a reference to the modified string *str1*.

friend ostream& operator<< (**ostream&** *os*, **const String&** *str*);
Overloads the output operator for a reference to a string object *str*.

inline friend ostream& operator<< (**ostream&** *os*, **const String*** *str*);
Overloads the output operator for a pointer to a string object *str*.

friend String& right_trim (**String&** *str1*, **const char*** *str2*);
Removes any suffix occurrence of the character string *str2* specified from the string object *str1*. This function returns a reference to the modified string *str2*.

inline Boolean operator< (**const char*** *str*) **const**;
Overloads the less-than operator for the **String** class. This function returns **TRUE** if the string is lexically less than the *char* s* argument; otherwise, this function returns **FALSE**.

inline Boolean operator< (**const String&** *str*) **const**;
Overloads the less-than operator for the **String** class. This function returns **TRUE** if the string object is lexically less than the string *str*; otherwise, this function returns **FALSE**.

inline Boolean operator<= (**const char*** *str*) **const**;
Overloads the less-than-or-equal operator for the **String** class. This function returns **TRUE** if the string object is lexically less than or equal to the character string argument *str*; otherwise, this function returns **FALSE**.

inline Boolean operator<= (**const String&** *str*) **const**;
Overloads the less-than-or-equal operator for the **String** class. This function returns **TRUE** if the string object is lexically less than or equal to the string *str*; otherwise, this function returns **FALSE**.

inline Boolean operator> (**const char*** *str*) **const**;
Overloads the greater-than operator for the **String** class. This function returns **TRUE** if the string object is lexically greater than the character string argument *str*; otherwise, this function returns **FALSE**.

inline Boolean operator> (**const String&** *str*) **const**;
Overloads the greater-than operator for the **String** class. This function returns **TRUE** if the string object is lexically greater than the string *str*; otherwise, this function returns **FALSE**.

inline Boolean operator>= (**const char*** *str*) **const**;
Overloads the greater-than-or-equal operator for the **String** class. This function returns **TRUE** if the string object is lexically greater than or equal to the character string argument *str*; otherwise, this function returns **FALSE**.

inline Boolean operator>= (**const String&** *str*) **const**;
Overloads the greater-than-or-equal operator for the **String** class. This function returns **TRUE** if the string object is lexically greater than or equal to the string *str*; otherwise, this function returns **FALSE**.

inline char operator[] (**long** *position*) **const**;
Returns the character at the zero-relative index *position* in the string. If the index is invalid, an Error exception is raised.

Boolean remove (**long** *start*, **long** *end*);
Removes the sequence of characters between the zero-relative inclusive *start* and exclusive *end* indexes. This function returns **TRUE** if successful; otherwise, this function returns **FALSE** if either one or both of the indexes is out of range.

Boolean replace (**const char*** *str*, **long** *start*, **long** *end*);
Replaces the sequence of characters between the zero-relative inclusive *start* and exclusive *end* indexes with a copy of the character string *str*. This function returns **TRUE** if successful; otherwise, this function returns **FALSE** if either one or both of the indexes is out of range.

void resize (**long** *size*);
Resizes the string object to hold at least *size* number of characters. If *size* is less than existing length, the operation is ignored.

String operator+ (**const char*** *str*);
Overloads the addition operator to concatenate a copy of the specified character sequence *str* to a string object. This function returns a new string object.

String operator+ (**const String&** *str*);
Overloads the addition operator to concatenate the value of the specified string object *str* to a string object. This function returns a new string object.

inline String& operator= (**char** *c*);
Overloads the assignment operator to assign a single character *c* to a string object. This function returns a reference to the modified string object.

inline String& operator= (**const char*** *str*);
Overloads the assignment operator to assign a copy of the specified character sequence *str* to a string object. This function returns a reference to the modified string object.

inline String& operator= (**const String&** *str*);
Overloads the assignment operator to assign the value of another string object *str* to a string object. This function returns a reference to the modified string object.

inline String& operator+= (**char** *c*);
Overloads the addition-and-assignment operator to concatenate a single character *c* to a string object. This function returns a reference to the modified string object.

inline String& operator+= (**const char*** *str*);
Overloads the addition-and-assignment operator to concatenate a copy of the specified character sequence *str* to a string object. This function returns a reference to the modified string object.

inline String& operator+= (**const String&** *str*);
Overloads the addition-and-assignment operator to concatenate the value of the specified string object *str* to a string object. This function returns a reference to the modified string object.

inline Boolean operator== (**const char*** *str*) **const**;
Overloads the equality operator for the **String** class. This function returns **TRUE** if the string object and *str* have the same sequence of characters; otherwise, this function returns **FALSE**.

inline Boolean operator== (**const String&** *str*) **const**;
Overloads the equality operator for the **String** class. This function returns **TRUE** if the strings have the same sequence of characters; otherwise, this function returns **FALSE**.

inline Boolean operator!= (**const char*** *str*) **const**;
Overloads the inequality operator for the **String** class. This function returns **FALSE** if the string object and *str* have the same sequence of characters; otherwise, this function returns **TRUE**.

inline Boolean operator!= (**const String&** *str*) **const**;
Overloads the inequality operator for the **String** class. This function returns **FALSE** if the strings have the same sequence of characters; otherwise, this function returns **TRUE**.

Name: **String** — Simple, dynamic string class

Synopsis: **#include** <COOL/String.h>

Base Classes: **Generic**

Friend Classes: None

Constructors:

String ();
Initializes an empty string object with a default size block of memory allocated to hold 100 characters.

String (**char** *c*);
Initializes a string object with the default size block of memory allocated to hold 100 characters whose value is the string consisting of the single character argument *c*.

String (**const char*** *str*);
Initializes a string object with the default size block of memory allocated to hold 100 characters whose value is copied from the specified character string argument *str*. If *str* is longer, the string will grow as necessary.

String (**const char*** *str*, **long** *size*);
Allocates an initial block of memory the size of the integer argument *size,* or the **strlen**(*str*), whichever is longer. and initializes the string object with a copy of the specified character string *str*. Note that *size* is ignored if less than the length of *str*.

String (**const String&** *str*);
Duplicates the size and value of another string object *str*.

String (**const String&** *str*, **long** *size*);
Duplicates the size and value of another string object *str* by allocating an initial block of memory to be the size of the integer argument *size,* or **strlen**(*str*), whichever is longer. The duplication is then performed.

Member Functions:

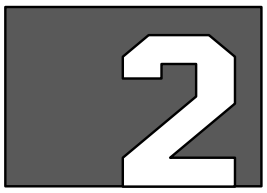
inline long capacity () **const**;
Returns the maximum number of characters that the string object can contain without having to grow.

void clear ();
Resets the **NULL** character string terminator to the beginning of the string and sets the length of the string to zero.

Boolean insert (**const char*** *str*, **long** *position*);
Inserts a copy of the sequence of characters pointed to by the first argument *str* at the zero-relative index provided by the second argument *position*. This function returns **TRUE** if successful; otherwise, this function returns **FALSE** if the index is out of range.

inline operator char* () **const**;
Provides an implicit conversion operator to convert a string object into a **char*** value.

String operator+ (**char** *c*);
Overloads the addition operator to concatenate a single character *c* to a string object. This function returns a new string object, via the stack.



STRING CLASSES

Introduction

2.1 The string classes are a collection of classes that implement textual operations and functions for such commonplace actions as string concatenation and growth, allocating memory as necessary and thus relieving the programmer from having to perform this task manually. The following classes and functions are discussed in this section:

- **String Class**
- **char*** functions
- **Regular Expression (Regexp) Class**
- **Gen_String Class**

The **String** class implements dynamic, efficient strings comparable to the built-in **char*** data type. The **char*** functions supplement the standard ANSI C character string library functions. Taken together, **String** and **char*** provide such operations as string concatenation, case sensitive and insensitive comparison, case conversion, and simple string search and replacement. The **Regexp** class provides a convenient mechanism to present regular expressions for complex pattern matching and replacement, and uses the built-in **char*** data type. The **Gen_String** class combines the functions of the **String** and **Regexp** classes, along with reference counting and garbage collection, to provide sophisticated character string manipulation.

Requirements

2.2 This section assumes you have a working knowledge of the C++ language and type system. In addition, you should understand the distinction between the concepts associated with overloaded operators, member functions, and friend functions.

String Class

2.3 The **String** class provides dynamic, efficient strings for a C++ application. The intent is to provide efficient **char***-like functionality that frees the programmer from worrying about memory allocation and deallocation problems, yet retains the speed and compactness of a standard **char*** implementation. The **String** class is dynamic in the sense that if an operation such as concatenate results in more characters than can fit in the currently allocated memory, the string object *grows* according to some established size or ratio value. All typical string operations are provided, including concatenation, case-sensitive and case-insensitive lexical comparison, string search, yank, delete, and replacement. System-provided functions for **char*** such as **strcpy** and **strcmp** are also available via the overloaded **operator char*** member function.

Printed on: Wed Apr 18 07:00:53 1990

Last saved on: Tue Apr 17 13:42:00 1990

Document: s2

For: skc