



Object-oriented Programming Strings

YoungWoon Cha
CSE Department
Spring 2023

Review

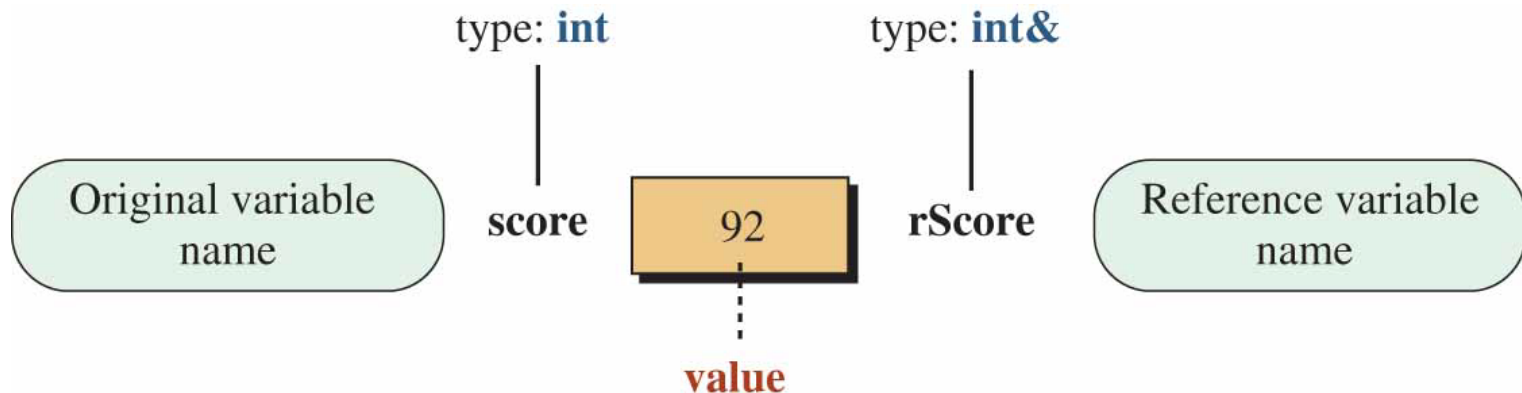
REFERENCES

A reference is an alternative name for an object.

References have been added to C++ to simplify communication between entities such as functions.

```
int score = 92;           // Declaring and initializing score
int& rScore = score;      // Declaring rScore and binding it to score
```

Figure 9.1 *Original variable and reference variable*



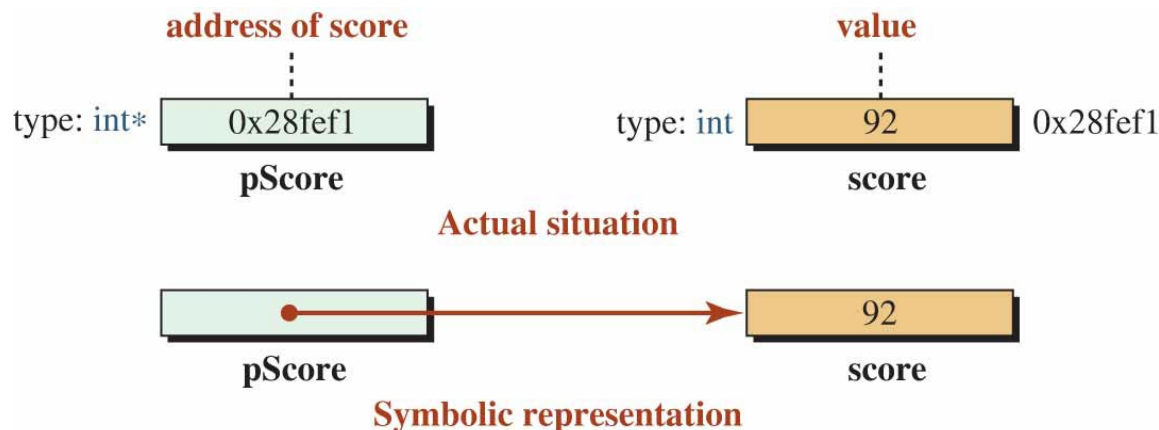
POINTERS

A *pointer* type is a compound type representing the address of a memory location.

A pointer variable is a variable whose contents are of pointer type.

In this section, we discuss addresses, pointer types, pointer variables and some related issues.

```
// Initialize pScore with address of score  
int* pScore = &score;
```

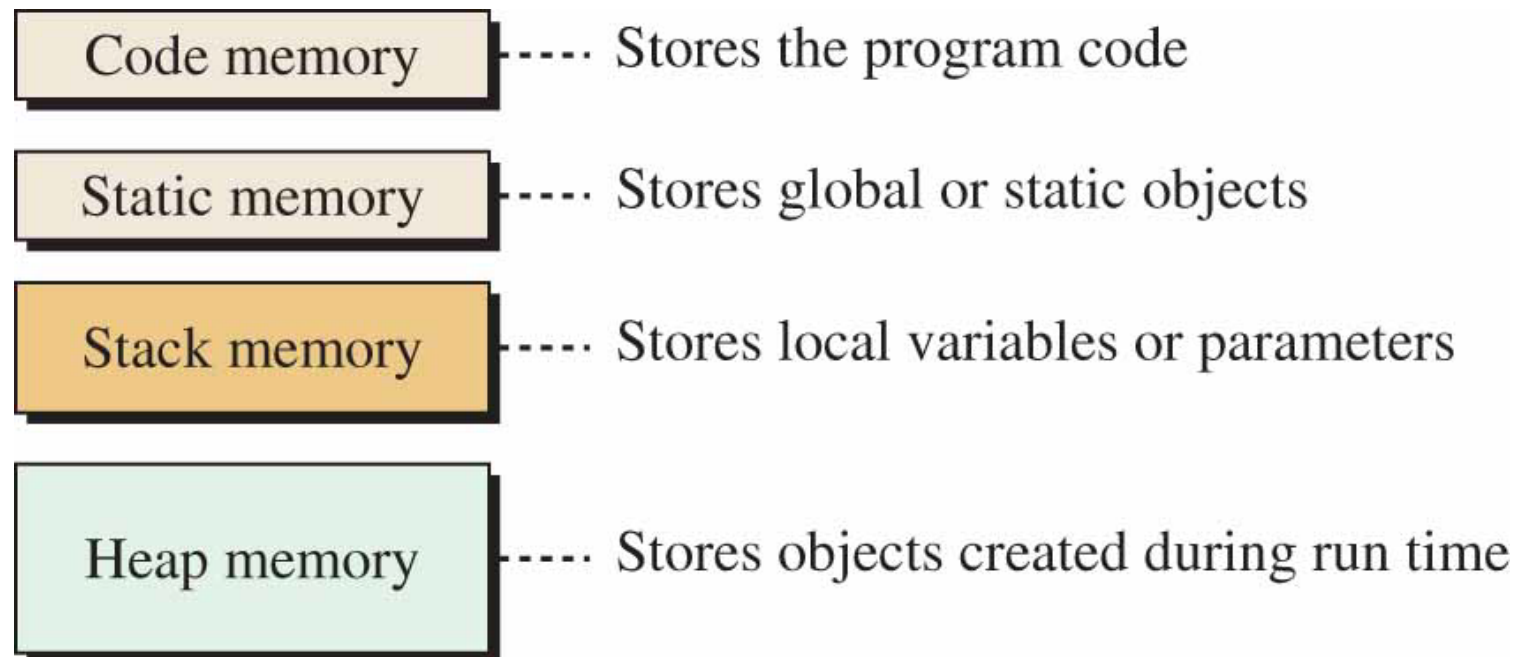


MEMORY MANAGEMENT

When a program in C++ is running, it uses memory locations.

The C++ environment divides memory into different areas.

Figure 9.22 *Memory sections used by a program*



C Strings

C-STRINGS Part 1

Although the C++ language contains the C++ string class type that we will discuss in the second section, we briefly discuss the C-string for two reasons.

First, there are some programs that still use C-strings.

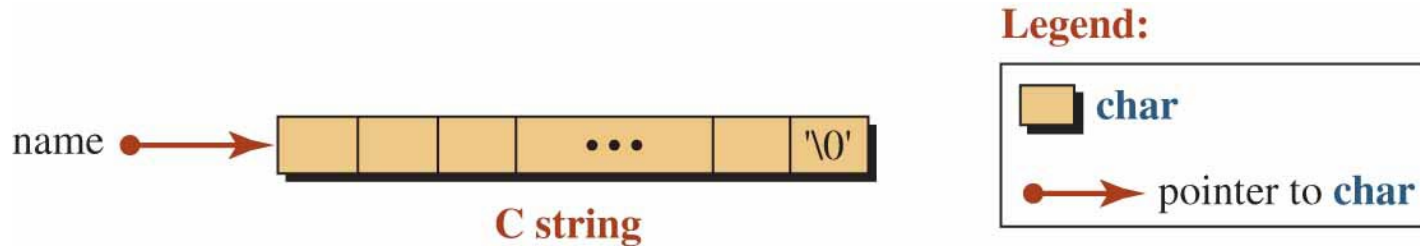
Second, the C++ string class uses some C-strings in its definitions.

The `<cstring>` header file is needed to use C-strings.

A C-string is not a class type. It is a null-terminated array of characters.

C-STRINGS Part 2

Figure 10.1 *General idea behind a C-string*



Since the name of an array is a pointer to the first element in the array, the name of a C-string is a pointer to the first character in the string.

However, we must remember that the name of a C-string does not define a variable; it defines a constant pointer.

The C-string name is a constant pointer to the first character.

Operations on C-Strings

A C-string is not a class type, which means there is no constructor defined in the library.

To construct a C-string we need to create an array of characters and set the last element to the null character `'\0'`.

We can create two types of C-strings: non-constant and constant.

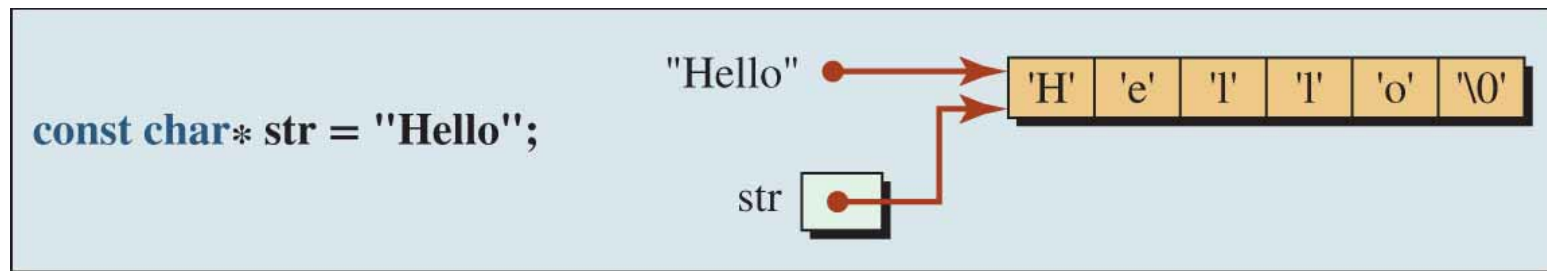
In a non-constant C-string, the value of the characters can be changed after creation; in a constant C-string, the value of the characters cannot be changed.

```
char str [ ] = {'A', 'B', 'C', 'D', '\0'};           // Non-constant
char str [ ] = "ABCD";                             // Non-constant compact
const char str [ ] = {'A', 'B', 'C', 'D', '\0'};    // Constant
const char str [ ] = "ABCD";                       // Constant compact
```

Operations on C-Strings

C++ forbids assigning a string literal to a non-constant pointer to character as shown below.

```
char* str = "Hello";           // Error. Literal is a constant
const char* str = "Hello";     // OK.
```



Pointer assignment by a literal

Compact Initializer and String Literal

We need to distinguish between a compact initializer and a string literal although both look the same.

```
char str1 [ ] = "Hello";       // "Hello" is a compact initializer
const char str2 [ ] = "Hello"; // "Hello" is a compact initializer
const char* str3 = "Hello";    // "Hello" is a literal string
```

C-Strings Construction in Heap Memory

Since a C-string is an array, we can create it in heap memory.

However, since the name of the string in this case is a pointer to character, we cannot use compact initialization.

If it is a non-constant string, we need to initialize it character by character; if it is a constant string, we need to use a string literal to do so.

```
char* str = new char [3]; // Non-constant string of two characters  
const char* str = new char [3]; // Constant string of two characters
```

C-Strings Destruction

If a C-string is created in stack memory, it is automatically deleted when the *main* function terminates.

When the string is created in the heap, we need to use the delete operator to delete it to avoid a memory leak.

```
const char* str = new char [3];    // Creation  
delete [ ] str;                    // Deletion
```

C-Strings Copying

The function *strcpy* replaces the first string with the whole of second string; the function *strncpy* replaces the first *n* characters in the first string with the first *n* characters of the second string.

Note that the copying changes the destination string, but not the source string.

```
strcpy(str1, str2);           // Using the whole str2  
strncpy(str1, str2, n);      // Using part of str2
```

The two functions *strcpy(...)* and *strncpy(...)* can be used to replace one string with another.

Copying in C-string Library Part 1

Program 10.1 Copying in a C string library

```
1  /*****
2  * The program shows how to use strcpy and strncpy to replace *
3  * the whole string or part of it with the whole or part of   *
4  * another string.                                           *
5  *****/
6  #include <cstring>
7  #include <iostream>
8  using namespace std;
9
10 int main ( )
11 {
12     // Copy the whole str2 to str1. String str1 is erased.
13     char str1 [] = "This is the first string.";
14     char str2 [] = "This is the second string.";
15     strcpy (str1, str2);
16     cout << "str1: " << str1 << endl;
17     // Copy part of str4 to str3. str3 is partially erased.
18     char str3 [] = "abcdefghijk.";
19     const char* str4 = "ABCDEFGHIJK";
20     strncpy (str3, str4, 4);
```

Copying in C-string Library Part 2

Program 10.1 Copying in a C string library

```
21 cout << "str3: " << str3 << endl;  
22 return 0;  
23 }
```

Run:

```
str1: This is the second string.  
str3: ABCDefghijk.
```

String Length Part 1

Each C-string has a size (length) that is the number of characters in the string without counting the null character.

The C-string library defines the *strlen* function to find the length of the string.

The function accepts a C-string and returns its length as a *size_t* type, which is defined as an *unsigned int* in the library.

```
size_t  n = strlen(str);           // Finding the length of str
```

The *strlen(...)* function returns the number of characters in a C-string not counting the null character.

String Length Part 2

Program 10.2 Finding the length of two strings

```
1  /*****
2  * The program shows how to get the length of a C-string, which *
3  * is the number of characters before the null character      *
4  *****/
5  #include <cstring>
6  #include <iostream>
7  using namespace std;
8
9  int main ( )
10 {
11     // Declaration and definition of four strings
12     const char* str1 = "Hello my friends.";
13     char str2 [] = {'H', 'e', 'l', 'l', 'o', '\0'} ;
14     // Finding and printing the length of each string
15     cout << "Length of str1: " << strlen (str1) << endl;
16     cout << "Length of str2: " << strlen (str2);
17     return 0;
18 }
```

Run:

Length of str1: 17

Length of str2: 5

C-Strings Input and Output

Overloaded Extraction and Insertion Operators

The `<cstring>` library has overloaded the extraction operator (`>>`) and the insertion operator (`<<`) to be used for string input and output.

The problem that we need to be aware of is that the character array that stores the input needs to have allocated enough memory locations to store all characters entered (before a whitespace) plus one for the null character.

```
cin >> str;           // Input
cout << str;           // Output
```

C-Strings Input and Output

Program 10.3 Using C strings

```
1  /*****
2  * The program shows how to create C-strings and use input and *
3  * output operations with them.                                *
4  *****/
5  #include <iostream>
6  using namespace std;
7
8  int main ( )
9  {
10     // Create one constant and one non-constant string
11     char str1 [] = {'H', 'e', 'l', 'l', 'o', '\0'} ;
12     const char str2 [] = {'H', 'e', 'l', 'l', 'o', '\0'};
13     // Create two constant string types and use string literals
14     const char* str3 = "Goodbye";
15     const char* str4 = "Goodbye\0 my friend";
16     // Printing four strings
17     cout << "str1: " << str1 << endl;
18     cout << "str2: " << str2 << endl;
19     cout << "str3: " << str3 << endl;
20     cout << "str4: " << str4 << endl << endl;
```

C-Strings Input and Output

Program 10.3 Using C strings

```
21 // Create and input a fifth string
22 char str5 [20];
23 cout << "Enter the characters for str5: " ;
24 cin >> str5;
25 cout << "str5: " << str5;
26 return 0;
27 }
```

Run:

```
str1: Hello
str2: Hello
str3: Goodbye
str4: Goodbye
```

```
Enter the characters for str5: This is the one.
str5: This
```

C-Strings the *getline* Function Part 1

To read a line of characters including whitespace, we need to use the function defined for this purpose, the *getline* function.

The *getline* function is a member of the *istream* class which means that we need to have an object of type *cin* to use it.

If the *delim* parameter is missing, the `'\n'` character is used.

```
in.getline(str, n);  
cin.get(str, n, '\n');
```

```
// Using '\n' as the delimiter  
// Using a specific delimiter
```

C-Strings the getline Function Part 2₍₁₎

Program 10.4 Using an array of strings

```
1  /*****
2   * The program shows how to read a set of lines using the      *
3   * getline function and print them.                             *
4   *****/
5  #include <iostream>
6  #include <cstring>
7  using namespace std;
8
9  int main ()
10 {
11     // Declaration of an array of strings
12     char lines [3][80];
13     // Inputting three lines
14     for (int i = 0; i < 3; i++)
15     {
16         cout << "Enter a line of characters: ";
17         cin.getline (lines [i], 80);
18     }
19     // Outputting three lines
20     cout << endl;
```

C-Strings the getline Function Part 2₍₂₎

Program 10.4 Using an array of strings

```
21     cout << "Output: " << endl;  
22     for (int i = 0; i < 3; i++)  
23     {  
24         cout << lines [i] << endl;  
25     }  
26     return 0;  
27 }
```

Run:

Enter a line of characters: This is the first line.
Enter a line of characters: This is the second line.
Enter a line of characters: This is the third line.

Output:

This is the first line.
This is the second line.
This is the third line.

C-Strings Accessing Characters Part 1

We can use the subscript operator to access a character if we know its position in the string.

Accessing means only retrieving if the string is a constant string.

Accessing can mean retrieving or changing if the string is a non-constant string.

```
char c = str [i];    // The string str is a constant  
str[i] = c;          // The string str is non-constant
```


C-Strings Accessing Characters Part 2

Program 10.5 Access characters in a C string

```
1  /*****
2   * The program shows how to access a character in a string      *
3   * using the subscript operator.                                *
4   *****/
5  #include <cstring>
6  #include <iostream>
7  using namespace std;
8
9  int main ( )
10 {
11     // Creation of two C-strings
12     const char* str1 = "Hello my friends.";
13     char str2 [ ] = "This is the second string.";
14     // Retrieving character at a given position
15     cout << "Character at index 6 in str1: " << str1[6] << endl;
16     // Changing character at a given position
17     str2 [0] = 't' ;
18     cout << "str2 after change: " << str2;
19     return 0;
20 }
```

C-Strings Accessing Characters Part 3

Program 10.5 *Access characters in a C string*

Run:

```
Character at index 6 in str1: m  
str2 after change: this is the second string.
```

C-Strings Searching for a Character Part 1

We can search a string to find a character. The search can return a pointer of the first occurrence (forward search) or the last occurrence (backward search).

The first search uses the *strchr* function; the second uses the *strrchr* function.

After a pointer to the character is set we can use it to change the character if the string is not constant.

If the character cannot be found, a *null* pointer is returned.

```
char* ptr = strchr(str, 'c');    // Forward search  
char* ptr = strrchr(str, 'c');  // Backward search
```

We can use *strchr(...)* and *strrchr(...)* member functions to create a pointer to a character.

C-Strings Searching for a Character Part 2

Program 10.6 Finding the start of a substring

```
1  /*****
2  * A program to search for a given character using forward      *
3  * search to find the first occurrence or backward search to    *
4  * to find the last occurrence.                                  *
5  *****/
6  #include <cstring>
7  #include <iostream>
8  using namespace std;
9
10 int main ( )
11 {
12     // Declaration of a string
13     char str [ ] = "Hello friends.";
14     // Capitalizing the first occurrence of character e
15     char* cPtr = strchr (str, 'e');
16     *cPtr = 'E' ;
17     cout << "str after first change: " << str << endl;
18     // Capitalizing the last occurrence of character e
19     cPtr = strrchr (str, 'e');
20     *cPtr = 'E' ;
```

C-Strings Searching for a Character Part 3

Program 10.6 *Finding the start of a substring*

```
21     cout << "str after last change: " << str << endl;  
22     return 0;  
23 }
```

Run:

```
str after first change: HEllO friends.  
str after last change: HEllO friEnds.
```

C-Strings Searching for a Substring Part 1

We can search a string to find the position of a substring using the `strstr` function.

The function returns a pointer to the first character in the substring.

If the substring cannot be found in the string, a *null* pointer is returned.

```
char* ptr = strstr(str, substr);    // Searching for a substring
```

C-Strings Searching for a Substring Part 2

Program 10.7 Finding a substring

```
1  /*****
2  * The program shows how to use the strstr member function to *
3  * find the occurrence of a substring in a string.           *
4  *****/
5  #include <cstring>
6  #include <iostream>
7  using namespace std;
8
9  int main ( )
10 {
11     // Creating a string
12     char str [ ] = "Hello friends of mine.";
13     // Finding the location of the substring
14     char* sPtr = strstr (str, "friends");
15     cout << "The substring starts at index: " << sPtr - str;
16     return 0;
17 }
```

Run:

The substring starts at index: 6

C-Strings Comparing

We can compare two strings using the *strcmp* and *strncmp* functions.

The first compares the two strings; the second compares only the first *n* characters of two strings.

Comparison is done character by character until a character that is not the same is reached (note that the null character is also used in the comparison).

When unequal characters are found, the comparison stops and the functions returns a negative number if the character in the first string is smaller than the second.

It returns a positive integer if the character in the first string is larger than the second. If different characters were not found, the functions return 0.

```
int value = strcmp(str1, str2);    // Comparing whole strings
int value = strncmp(str1, str2, n); // Comparing n characters
```

Two strings can be compared with *strcmp(...)* and *strncmp(...)* member functions.

C-Strings Comparing

Program 10.9 Comparing C strings

```
1  /*****
2   * The program shows how we can compare two strings using the *
3   * strcmp and strncmp functions                                *
4   *****/
5  #include <cstring>
6  #include <iostream>
7  using namespace std;
8
9  int main ( )
10 {
11     // Declaration of two C-strings
12     const char* str1 = "Hello Alice.";
13     const char* str2 = "Hello John.";
14     const char* str3 = "Hello Betsy.";
15     // Comparison use the whole length
16     cout << "Comparing str1 and str2: ";
17     cout << strcmp (str1, str2) << endl;
18     cout << "Comparing str2 and str3: ";
19     cout << strcmp (str2, str3) << endl;
20     // Comparison using one the first characters
```

C-Strings Comparing

Program 10.9 Comparing C strings

```
21     cout << "Comparing first 5 characters of str1 and str2: ";  
22     cout << strcmp (str1, str2, 5);  
23     return 0;  
24 }
```

Run:

Comparing str1 and str2: -1

Comparing str2 and str3: 1

Comparing first 5 characters of str1 and str2: 0

C-Strings Concatenation (Appending)

We can add a string at the end of another string.

In this case the destination string will be changed, but the source remains unchanged.

The library defines two member functions for this purpose: the `strcat` and `strncat` functions.

The first function concatenates all characters in *str2* at the end of *str1*.

The second function concatenates only the first *n* characters of the second string at the end of the first string.

However, we need to be sure that the *str1* has enough memory allocation to accept the concatenation (Program 10-10).

```
strcat(str1, str2);           // Appending the whole string
strncat(str1, str2, n);       // Appending the first n characters
```

The member functions `strcat(...)` and `strncat(...)` can be used to concatenate one string at the end of the other.

C-Strings Concatenation(Appending)

Program 10.10 Using strcat and strncat

```
1  /*****
2   * We use strcat and strncat to concatenate a string at the *
3   * end of another string. *
4   *****/
5  #include <cstring>
6  #include <iostream>
7  using namespace std;
8
9  int main ( )
10 {
11     // Using strcat function
12     char str1 [20] = "This is ";
13     const char* str2 = "a string.";
14     strcat (str1, str2);
15     cout << "str1: " << str1 << endl;
16     // Using strncat function
17     char str3 [20] = "abcdefghijk";
18     const char* str4 = "ABCDEFGHIJK";
19     strncat (str3, str4, 4);
20     cout << "str3: " << str3 << endl;
```

C-Strings Concatenation(Appending)

Program 10.10 *Using strcat and strncat*

```
21     return 0;  
22 }
```

Run:

```
str1: This is a string.  
str3: abcdefghijkABCD
```

C-Strings Tokenizing

One of the common operations on a string is to find tokens embedded in a string.

Tokens are substrings separated by delimiters (such as whitespace).

The library defines the *strtok* function.

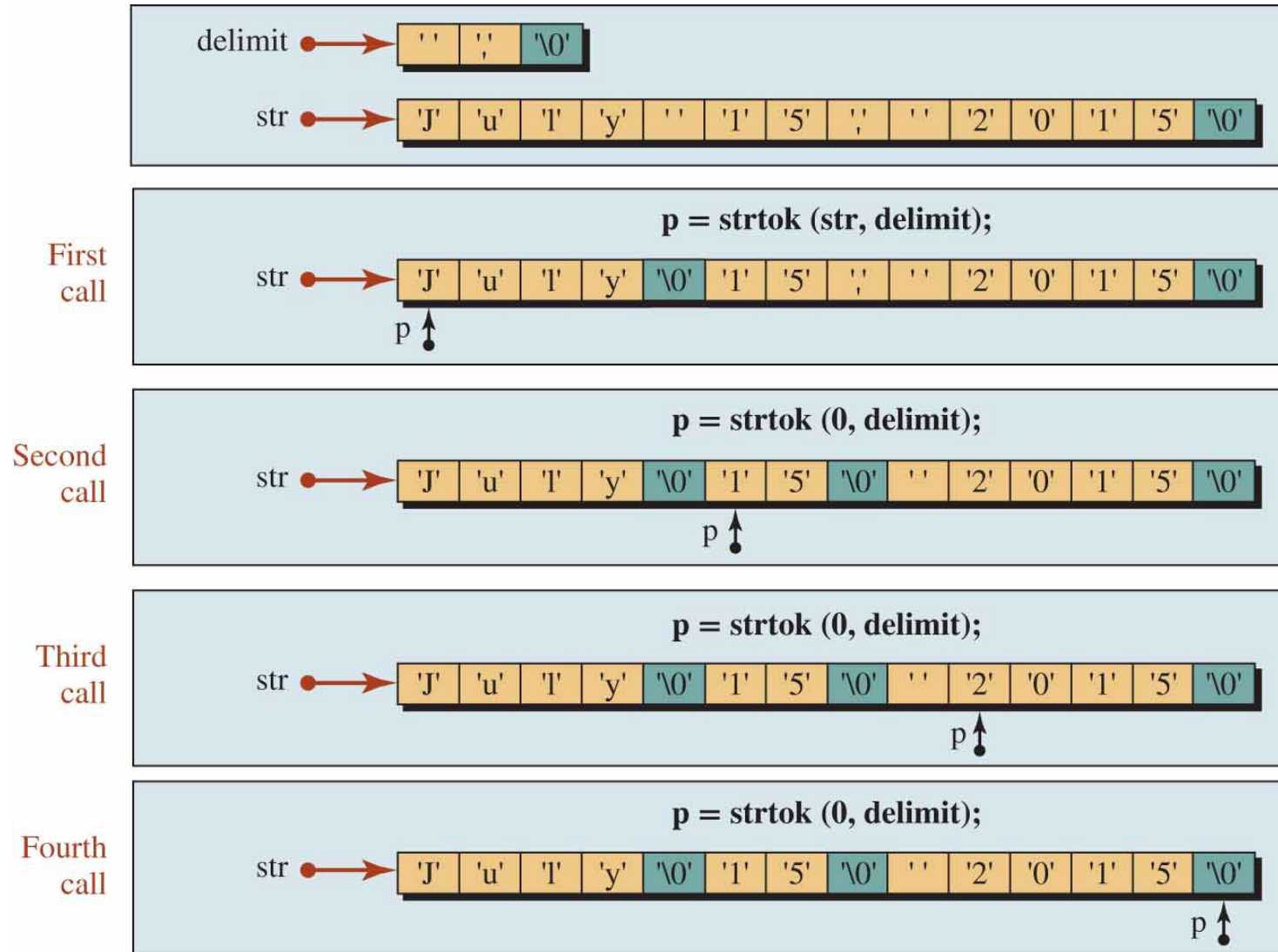
To find all tokens in a string, we need to call the *strtok* function multiple times.

```
char* p = strtok(str, delimiter); // Tokenizing using a delimiter
```

The *strtok(...)* function can be used to split a string into tokens using delimiter characters.

C-Strings Tokenizing

Figure 10.6 *Tokenizing using space and comma as delimiter*



C-Strings Tokenizing

Program 10.11 Tokenizing a string

```
1  /*****
2   * The program shows how we can use strtok function to extract *
3   * tokens from a date.                                         *
4   *****/
5  #include <cstring>
6  #include <iostream>
7  using namespace std;
8
9  int main ( )
10 {
11     // Declaration of a sentence and a pointer
12     char str [ ] = "July 15, 2015";
13     char* p;
14     // Use strtok to extract all words
15     p = strtok (str, ", "); // first call
16     while (p)
17     {
18         cout << p << endl;
19         p = strtok (0, ", "); // second, third, and fourth calls
20     }
```


C-Strings Tokenizing

Program 10.11 *Tokenizing a string*

```
21     return 0;  
22 }
```

Run:

July
15
2015

Although C-strings provide a way to use strings, they are more error-prone and not as robust as C++ strings.

We therefore recommend C++ strings be used as much as possible.

C++ Strings

THE C++ STRING CLASS

The C++ library provides a class named *string*, whose objects are normally referred to as C++ string objects as compared to C-string arrays of characters.

To use the objects and member functions of this class, we need to include the header file `<string>` in our program.

We need the `<string>` header file to use C++ strings.

General Design Idea

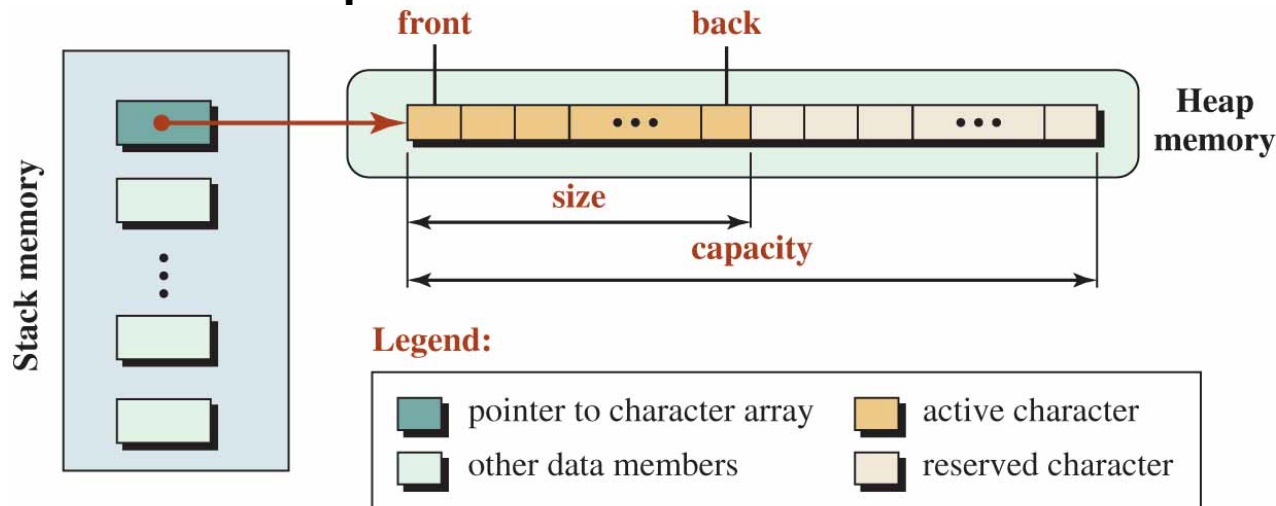
The string class has private data members and public member functions.

The user calls the public member functions to manipulate string objects.

In general, the data members include a pointer to an array of characters.

Other data members keep information about the character array as we discuss gradually.

The data members are normally created in stack memory, but the character array itself is allocated in the heap because its size is not defined until run time.



Notes:

1. A C++ string is not null terminated.
2. Size must be less than or equal to capacity.
3. The front is at index 0.
4. The back is at index (size - 1).

A C++ string is an array of characters, but it is not null-terminated.

C++ String Library Part 1

Table 10.2 *Part of C++ library*

// Constructors

```
string :: string ()  
string :: string (size_type count, char c)  
string :: string (const char* cstr)  
string :: string (const char* cstr, size_type count)
```

// Destructor

```
string :: ~string()
```

// Copy constructors

```
string :: string (const string& strg)  
string :: string (const string& strg, size_type index, size_type length = npos)
```

// Operations related to size and capacity

```
size_type string :: size ( ) const  
size_type string :: max_size ( ) const  
void string :: resize (size_type n, char c )  
size_type string :: capacity ( ) const  
void string :: reserve (size_type n = 0)  
bool string :: empty ( ) const
```

C++ String Library Part 2

Table 10.2 *Continued*

// Input and output

```
istream& operator>> (istream& in, string& strg)
ostream& operator<< (ostream& out, const string& strg)
istream& getline (istream& in, string& strg)
istream& getline (istream& in, string& strg, char delimit)
```

// Accessing a character given its position

```
const char& string::operator[ ] (size_type pos) const
char& string::operator[ ] (size_type pos)
const char& string::at (size_type pos) const
char& string::at (size_type pos)
```

// Accessing a substring given the position of the first character and length

```
string string::substr (size_type pos = 0, size_type length = npos) const
```

// Finding the position of a given character (forward or backward search)

```
size_type string::find (char c, size_type index = 0) const
size_type string::rfind (char c, size_type index = npos) const
```

C++ String Library Part 3

Table 10.2 *Continued*

// Finding the position of a character in a set (forward or backward search)

size_type string :: find_first_of (const string& temp, size_type pos = 0)

size_type string :: find_last_of (const string& temp, size_type pos = npos)

// Finding the position of a character not in a set (forward or backward search)

size_type string :: find_first_not_of (const string& temp, size_type pos = 0)

size_type string :: find_last_not_of (const string& temp, size_type pos = npos)

// Comparing two strings

int string :: compare (size_type pos1, size_type n1, const string strg2,
size_type pos2, size_type n2) const

int string :: compare (size_type pos1, size_type n1,
const char* cstr, size_type n2) const

// Logical comparison of two strings (oper can be <, <=, >, >=, ==, !=)

bool string :: operator**Oper** (const string strg1, const string strg2)

bool string :: operator**Oper** (const string strg1, const char* cstr)

bool string :: operator**Oper** (const char* cstr, const string strg1)

C++ String Library Part 4

Table 10.2 *Continued*

// Modifying a string using another string (append, insert, replace, and assign)

string& string::append (const string& temp)

string& string::insert (size_type pos, const string& temp)

string& string::replace (size_type pos, size_type n, const string& temp)

string& string::assign (size_type pos, size_type n, const string& temp)

// Clearing and erasing a string

void string::clear ()

string& string::erase (size_type pos = 0, size_type n = npos)

// Using the assignment operator

string& string::operator= (const string& strg)

string& string::operator= (const char* cstr)

string& string::operator= (char c)

// Pushing a character at the end of a string

void string::push_back (char c)

C++ String Library Part 5

Table 10.2 *Continued*

// Using the compound assignment (addition)

string& string :: operator+= (const string& strg)

string& string :: operator+= (const char* cstr)

string& string :: operator+= (char c)

// Using the addition operator

string& string :: operator+ (const string& strg1, const string& strg2)

string& string :: operator+ (const string& strg1, const char* cstr2)

string& string :: operator+ (const char* cstr1, const string& strg2)

string& string :: operator+ (const string& strg1, char c)

// Conversion to a character array

const char* string :: data () const

// Conversion to a C-string

const char* string :: c_str () const

C++ String Construction

The C++ string defines one default constructor and three parameter constructors.

Default Constructor

The default constructor creates an empty string by setting the pointer data member to 0.

```
string strg;    // Creating an empty string object
```

Parameter Constructors

We can use a set of characters of the same value, a string literal, or a part of a string literal.

```
string strg1(5 , 'a');    // The string "aaaaa"  
string strg2("hello");    // The string "hello"  
string strg3("hello", 2);    // The string "he"
```

C++ String *Construction and Destruction*

Destruction

The destructor of the string class simply deletes the character array created in the heap and pops all data members allocated in the stack.

Copy Construction

The string class allows us to use two different copy constructors.

We know that a copy constructor needs to use an existing object.

However, we can have a copy constructor made of a full existing object or part of an existing object.

```
string strg(oldStrg); // Using the whole oldStrg
string strg(oldStrg, index, length ); // Using part of the oldStrg
```

Size and Capacity

A C++ string object uses an array of characters in the heap. If the size of the array needs to be decreased during an operation, the value of the size member function is changed.

Size and Maximum Size

There are two functions that return a value about the string size.

```
size_type n = strg.size();           // Getting the size
size_type n = strg.length();         // Getting the size
size_type n = strg.max_size();       // Getting the maximum size
```

C++ String Size

Capacity and Reserve

The capacity function returns the current capacity of the character array.

We can call the reserve function to make the capacity larger than the size.

```
size_type n = strg.capacity();    // Getting the capacity
strg.reserve(n);                  // Reserving a larger array
```

Emptiness

The empty function returns *true* if the size is 0; *false* otherwise.

```
bool fact = strg.empty();         // Checking emptiness
```

C++ String Size

Program 10.12 *Testing functions related to size and capacity*

```
1  /*****
2  * The program creates a string object and then tests the size,*
3  * maximum size, and capacity before and after reservation.    *
4  *****/
5  #include <string>
6  #include <iostream>
7  using namespace std;
8
9  int main ( )
10 {
11     // Creating a string object
12     string strg ("Hello my friends");
13     // Test size, maximum size and capacity
14     cout << "Size: " << strg.size () << endl;
15     cout << "Maximum size: " << strg.max_size() << endl;
16     cout << "Capacity: " << strg.capacity() << endl;
17     cout << "Empty? " << boolalpha << strg.empty() << endl;
18     cout << endl;
19     // Making reservation and test again
20     strg.reserve (20);
```

C++ String Size

Program 10.12 *Testing functions related to size and capacity*

```
21     cout << "Size: " << strg.size () << endl;  
22     cout << "Maximum size: " << strg.max_size() << endl;  
23     cout << "Capacity: " << strg.capacity() << endl;  
24     cout << "Empty? " << boolalpha << strg.empty();  
25     return 0;  
26 }
```

Run:

```
Size: 16  
Maximum size: 1073741820  
Capacity: 16  
Empty? false
```

```
Size: 16  
Maximum size: 1073741820  
Capacity: 32  
Empty? false
```


C++ String Input and Output

Program 10.13 *Using input/output operators*

```
1  /*****
2   * A program to test input/output operators with string objects. *
3   *****/
4  #include <string>
5  #include <iostream>
6  using namespace std;
7
8  int main ( )
9  {
10     // Constructing a default object
11     string strg;
12     // Inputting and outputting values for the strg object
13     cout << "Input the string: " ;
14     cin >> strg;
15     cout << strg << endl;
16     return 0;
17 }
```

C++ String Input and Output

Program 10.13 *Using input/output operators*

Run:

```
Input the string: Hello  
Hello
```

Run:

```
Input the string: Hi my friends  
Hi
```

C++ String Input and Output

The getline Function

To allow the user to have more control, the *istream* object has a function named *getline* with two versions.

The first version uses `'\n'` as the delimiter, which means it can read the whole line; the second version allows users to define their own delimiter character.

```
getline(in, strg);           // The input stops with '\n'  
getline(in, strg, 'c');      // The input stops with character c
```

C++ String Input and Output

Program 10.14 Using getline for input

```
1  /*****
2   * A program to test the getline function with strings.      *
3   *****/
4  #include <string>
5  #include <iostream>
6  using namespace std;
7
8  int main ( )
9  {
10     // Constructing a default object
11     string strg;
12     // Creating a string made of a single line
13     cout << "Enter a line of characters: " << endl;
14     getline (cin, strg);
15     cout << strg << endl << endl;
16     // Creating a string made of multiple lines
17     cout << "Enter lines of characters ended with $: " << endl;
18     getline (cin, strg, '$');
19     cout << strg;
20     return 0;
```

C++ String Input and Output

Program 10.14 *Using getline for input*

```
21 }
```

Run:

Enter a line of characters:

This is a line of text.

This is a line of text.

Enter lines of characters ended with \$:

**This is a multi-line set of
characters to be
stored in a string.\$**

This is a multi-line set of
characters to be
stored in a string.

C++ String Accessing Characters

We can access an individual character to retrieve or change it if we know its index (its positions in the string relative to zero).

The string class provides four member functions for this purpose.

The first two use the subscript operator [] to return a character as an *rvalue* or *lvalue*.

The second two members use the *at* function to select a character.

```
char c = strg [pos];           // character c can be modified
char c = strg.at(pos);         // character c can be modified
const char c = strg [pos];     // character c cannot be modified
const char c = strg.at(pos);   // character c cannot be modified
```

We use accessing character functions to provide the position of the character to be returned.

C++ String Accessing Characters

Program 10.15 Retrieving and changing characters

```
1  /*****
2  * The program shows how to retrieve a single character in a *
3  * string. *
4  *****/
5  #include <string>
6  #include <iostream>
7  using namespace std;
8
9  int main ( )
10 {
11     // Construction of a string
12     string strg ("A short string");
13     // Retrieving and printing characters at index 5 and 8
14     cout << "Character at index 5: " << strg [5] << endl ;;
15     cout << "Character at index 8: " << strg.at(8) << endl;
16     return 0;
17 }
```

Run:

```
Character at index 5: r
Character at index 8: s
```

C++ String Accessing Characters

Program 10.16 *Changing all characters to uppercase*

```
1  /*****
2  * The program shows how we can capitalize a line of text using *
3  * the operator [] as an lvalue and rvalue.                      *
4  *****/
5  #include <string>
6  #include <iostream>
7  using namespace std;
8
9  int main ( )
10 {
11     string line;
12
13     cout << "Enter a line of text: " << endl;
14     getline (cin, line);
15     for (int i = 0; i < line.size(); i++)
16     {
17         line[i] = toupper (line[i]);
18     }
19     cout << line;
20     return 0;
```


C++ String Accessing Characters

Program 10.16 *Changing all characters to uppercase*

```
21 }
```

Run:

Enter a line of text:

This is a line of text to be capitalized.

THIS IS A LINE OF TEXT TO BE CAPITALIZED.

C++ String Accessing Characters

Program 10.17 Reversing a string object

```
1  /*****
2   * The program uses a function to reverse a string object.      *
3   *****/
4  #include <string>
5  #include <iostream>
6  using namespace std;
7
8  void reverse (string& strg); // Function declaration
9
10 int main ( )
11 {
12     // Declaration of string object
13     string strg;
14     // Input the original object and print it
15     cout << "Enter a string: ";
16     getline (cin, strg);
17     cout << "Original string: " << strg << endl;
18     // Reverse the object and print it
19     reverse (strg);
20     cout << "Reversed string: " << strg;
```

C++ String Accessing Characters

Program 10.17 Reversing a string object

```
21     return 0;
22 }
23 /*****
24  * The function reverses a string passed by reference to it.  *
25  *****/
26 void reverse (string& strg)
27 {
28     string temp (strg);
29     int size = strg.size () ;
30     for (int i = 0; i < size; i++)
31     {
32         strg [i] = temp [size - 1 - i];
33     }
34 }
```

Run:

Enter a string: **Hello my friends.**

Original string: Hello my friends.

Reversed string: .sdneirf ym olleH

C++ String Retrieving a Substring

We can retrieve a substring, a set of consecutive characters, from a string by giving the index of the first character and the number of characters to be retrieved (length).

Since only the left-most parameters can be set to default, if only one parameter is given it is taken as pos.

If both parameters are missing, the whole string is returned.

Note that the function is defined as constant, which means that the host object can not be changed.

In the following result is a string that is created in the process.

```
string result = strg.substr(pos, n); // result has n characters
```

C++ String Retrieving a Substring

Program 10.18 Retrieving two substrings

```
1  /*****
2  * The program shows how to retrieve two substrings from a      *
3  * string object                                              *
4  *****/
5  #include <string>
6  #include <iostream>
7  using namespace std;
8
9  int main ( )
10 {
11     // Construction of a string
12     string strg ("The C++ language is fun to work with.");
13     // Retrieving two substrings.
14     cout << strg.substr(8) << endl ;
15     cout << strg.substr(4,12) << endl;
16     return 0;
17 }
```

Run:

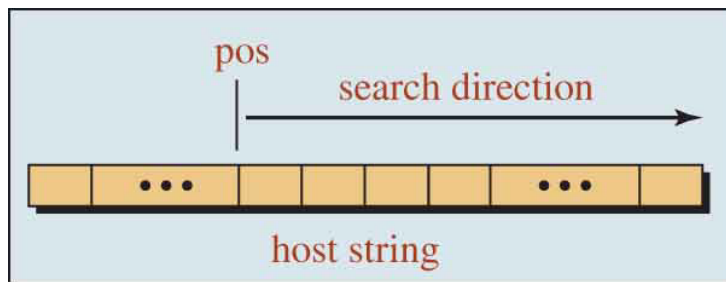
```
language is fun to work with.
C++ language
```

Searching for A Character

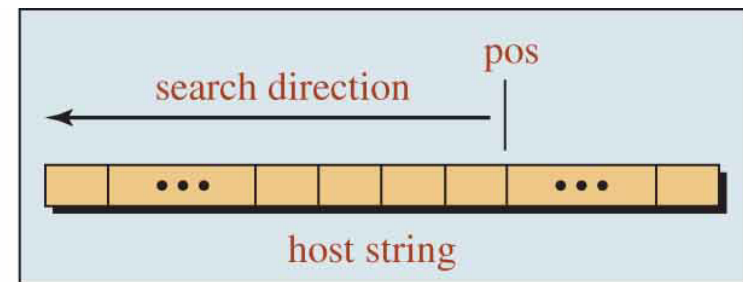
Forward and Backward Search for a given Character

We can use two member functions (*find* and *rfind*) to search for a specific character

Figure 10.8 *Search for a character in a string*



forward search



backward search

Note:

If the character is not found (in either direction), the constant *npos* (−1) is returned.

```
size_type pos = strg.find(c, index);    // forward search
size_type pos = strg.rfind(c, index);   // backward search
```

Searching for A Character

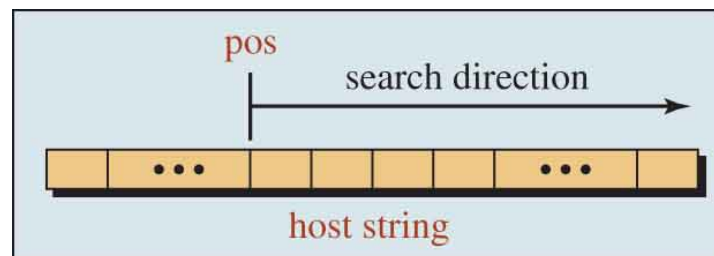
Forward or Backward Search for A Character Belonging to a Set

A more interesting search is when need to find any character in a set.

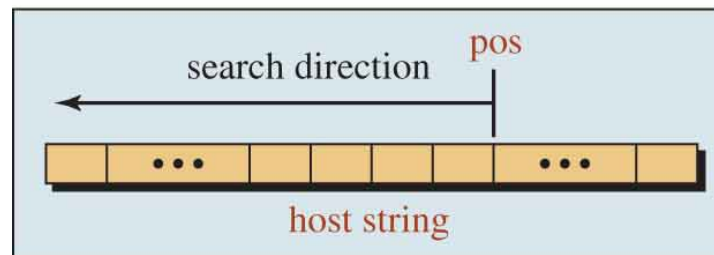
Another alternative is that we want to look for a character not in a set.

To make the format of the functions easier to understand, *we create a temporary host object defining the set.*

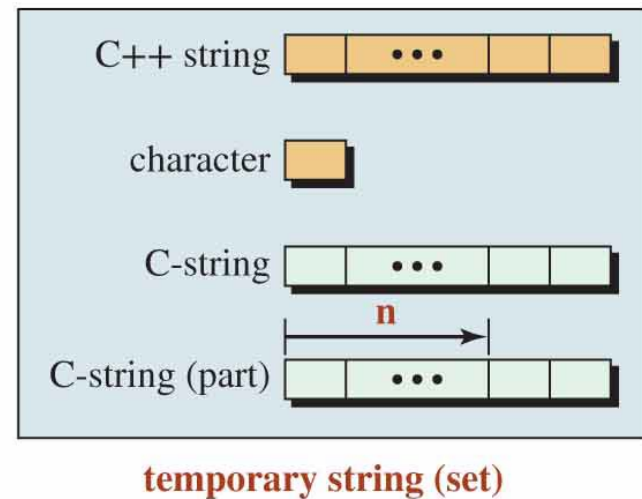
Figure 10.9 *The idea behind four find functions*



find_first_of and *find_first_not_of*



find_last_of and *find_last_not_of*



Searching for A Character₁

Program 10.19 *Retrieving words from a line of text*

```
1  /*****
2   * The program uses search functions to find and extract words *
3   * in a line of text.                                         *
4   *****/
5  #include <string>
6  #include <iostream>
7  using namespace std;
8
9  int main ( )
10 {
11     // Declaration of variables, types, and constants
12     string text, word;
13     string delimiter (" \n");
14     string::size_type wStart, wEnd;
15     string :: size_type npos;
16     // Input a line of text from keyboard
17     cout << "Enter a line of text: " << endl ;
18     getline (cin, text);
19     // Search, find, and print words
20     cout << "Words in the text:" << endl;
```


Searching for A Character₂

Program 10.19 Retrieving words from a line of text

```
21     wStart = text.find_first_not_of (delimiter, 0);
22     npos = text.length();
23     while (wStart < npos)
24     {
25         wEnd = text.find_first_of (delimiter, wStart);
26         cout << text.substr (wStart, wEnd - wStart) << endl;
27         wStart = text.find_first_not_of (delimiter, wEnd);
28     }
29     return 0;
30 }
```

Run:

Enter a line of text:

This is a line of text.

Words in the text:

This

is

a

line

of

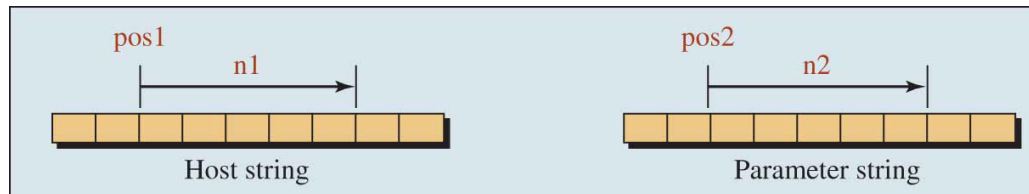
text.

Comparing Strings Integral Comparison

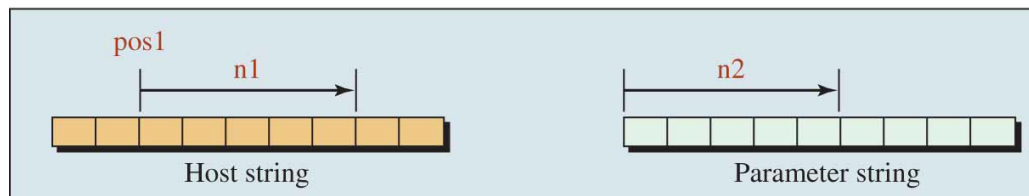
Integral Comparison

The integral comparison compares two strings and returns one of three integral values: zero when the two strings are equal, a positive number when the first string is greater than the second, and a negative number when the first string is less than the second.

```
// Two C++ strings
int result = strg.compare(pos1, n1, strg2, pos2, n2);
// A C++ string and a C string
int result = strg.compare(pos1, n1, str, n2);
```



Comparing two C++ strings

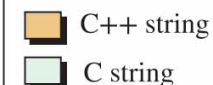


Comparing a C++ string and a C string

Note:

If pos1 or pos2 is missing, it means the beginning of the string.
If n1 or n2 is missing, it means the end of the string.

Legend:



Comparing Strings Using Integral Comparison Part 1

Program 10.20 *Integral comparison of strings*

```
1  /*****
2  * The program to test integral comparison.          *
3  *****/
4  #include <string>
5  #include <iostream>
6  using namespace std;
7
8  int main ( )
9  {
10     // Declaration of two C++ strings
11     string strg1 ("Hello my friends");
12     string strg2 ("Hello friends");
13     // Comparing two C++ strings
14     cout << strg1 << " compared with " << strg2 << ": ";
15     cout << strg1.compare (strg2) << endl;
16     // Comparing part of the two C++ strings
17     cout << "Hello compared with Hello: ";
18     cout << strg1.compare( 0, 5, strg2, 0, 5) << endl;
19     // Comparing part of the first C++ string and a C-string
20     cout << "Hello compared with Hello: ";
```

Comparing Strings Using Integral Comparison Part 2

Program 10.20 *Integral comparison of strings*

```
21     cout << strg1.compare (0, 5, strg2) << endl;  
22     // Comparing part of a C++ string and part of a C-string  
23     cout << "Hel compared with Hell: ";  
24     cout << strg2.compare (0, 3, "Hello" ,4);  
25     return 0;  
26 }
```

Run:

```
Hello my friends compared with Hello friends: 1  
Hello compared with Hello: 0  
Hello compared with Hello: -8  
Hel compared with Hell: -1
```

Comparing Strings Logical Comparison Part 1

The logical comparison compares two strings and returns a Boolean value (*true* or *false*).

Just like the integral comparison, two string are compared.

The first is always the host and the second is a C++ string or a C-string. Unlike the integral comparison, we cannot compare two substrings.

If we want to do so, we need to make temporary C++ strings out of the substrings and then compare them.

```
// Comparing two C++ strings
bool result = strg1 oper strg2;
// Comparing a C++ string and a C string
bool result = strg oper str;
// Comparing a C string and a C++ string
bool result = str oper strg;
```

Comparing Strings Logical Comparison Part 2

Program 10.21 *Using logical operators to compare strings*

```
1  /*****
2  * The program to test logical operators to compare two strings. *
3  *****/
4  #include <string>
5  #include <iostream>
6  using namespace std;
7
8  int main ( )
9  {
10     // Creation of four C++ strings
11     string strg1;
12     string strg2 (5, 'a');
13     string strg3 ("Hello Friends");
14     string strg4 ("Hi People", 4);
15     // Using six logical operators (relational and equality)
16     cout << "strg1 < strg2 : " << boolalpha << (strg1 < strg2);
17     cout << endl;
18     cout << "strg2 >= strg3: " << boolalpha << (strg2 >= strg3);
19     cout << endl;
20     cout << "strg1 = strg2: " << boolalpha << (strg1 = strg2);
```

Comparing Strings Logical Comparison Part 3

Program 10.21 *Using logical operators to compare strings*

```
21     cout << endl;  
22     cout << "Hi P != strg4: " << boolalpha << ("Hi P" != strg4);  
23     return 0;  
24 }
```

Run:

```
strg1 < strg2 : true  
strg2 >= strg3: true  
strg2 < strg3: false  
strg1 = strg2: false  
Hi P != strg4: false
```

Adding Character to a String

Push_Back

We often need to add a character to a string.

The string library defines the *push_back* function that adds a character at the end of the string.

```
strg.push_back(c);           // Append character c at the end of strg
```

Partial or Total Erasure

There are two functions that can totally or partially erase the characters in the string without destructing the string.

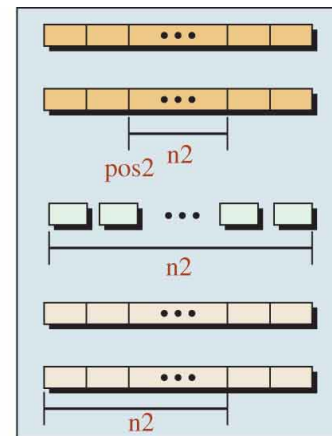
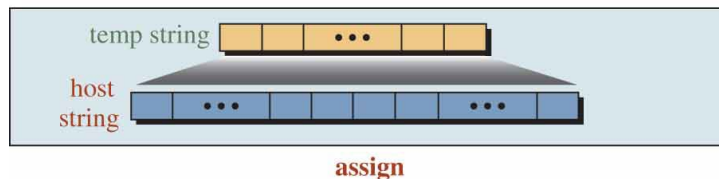
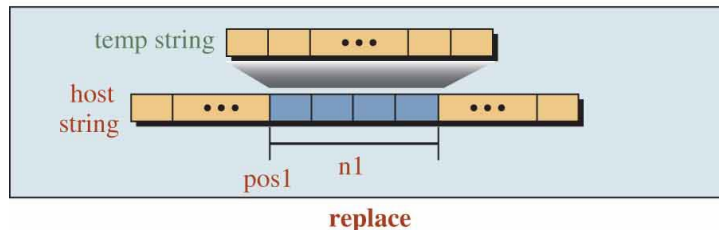
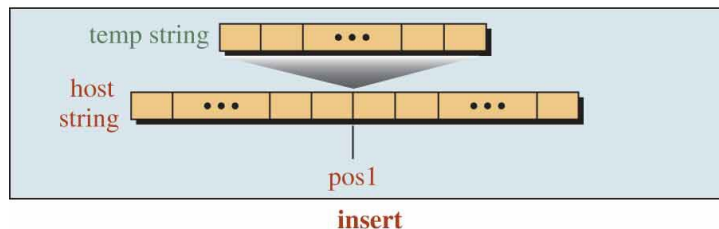
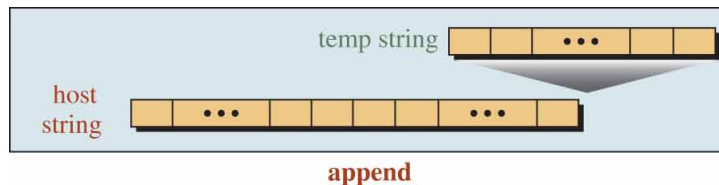
The object still exists, but partially or totally devoid of its characters.

```
strg.clear();                // Erase all character in the string  
strg.erase(pos, n);         // Erase part of the string
```


Adding Character to a String

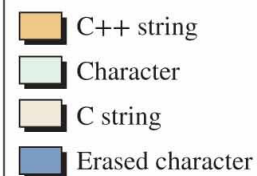
The following shows how we use these four category of functions:

```
strg.append(temp);           // Append
strg.insert(pos1, temp);     // Insert
strg.replace(pos1, n1, temp); // Replace
strg.assign(temp);           // Assign
```



Five ways to make a temporary string

Legend:



Overloaded Operators and Strings

Some of the modifying operations are achieved using overloaded operators.

C++ overloads the assignment operator (=), the compound assignment operator (+=) and the addition operator (+) as shown below.

The *temp*, *temp1*, and *temp2* are instances of the string class.

```
string strg = temp;           // Assignment
string strg += temp;          // Compound Assignment
string strg = temp1 + temp2;  // Addition
```

Print Full Name of a Person Part 1

Program 10.23 *Printing the full name of a person*

```
1  /*****
2   * A program to show concatenation of strings and characters.  *
3   *****/
4  #include <string>
5  #include <iostream>
6  using namespace std;
7
8  int main ( )
9  {
10     // Declarations
11     string first, last;
12     char init;
13     // Input first, last, and initial
14     cout << "Enter first name: ";
15     cin >> first;
16     cout << "Enter last name: ";
17     cin >> last;
18     cout << "Enter initial: ";
19     cin >> init;
20     // Printing the full name in one format
```

Print Full Name of a Person Part 2

Program 10.23 *Printing the full name of a person*

```
21     cout << endl;
22     cout << "Full name in first format: ";
23     cout << first + " " + init + "." + " " + last << endl << endl;
24     // Printing the full name in another format
25     cout << "Full name in second format: ";
26     cout << last + ", " + first + " " + init + ".";
27     return 0;
28 }
```

Run:

Enter first name: John

Enter last name: Brown

Enter initial: A

Full name in first format: John A. Brown

Full name in second format: Brown, John A.

C++ Strings and Conversion

Conversion

We can convert a C++ object to a character array or to a C-string.

```
const char* arr = strg.data();    // Conversion to a character array  
const char* str = strg.c_str();   // Conversion to a C-string
```

Example Problems

Conversion in Positional Number System

In computer science, we use different positional numbering system: *binary, octal, decimal, and hexadecimal*.


Each positional numbering system uses a set of symbols and a *base*.

The base defines the total number of symbols used in the system.

Table 10.3 shows the base and the symbols we work with in programming.

Note that the value of symbols A, B, C, D, E, F are 10, 11, 12, 13, 14, and 15 respectively.

Table 10.3 *Positional number system*



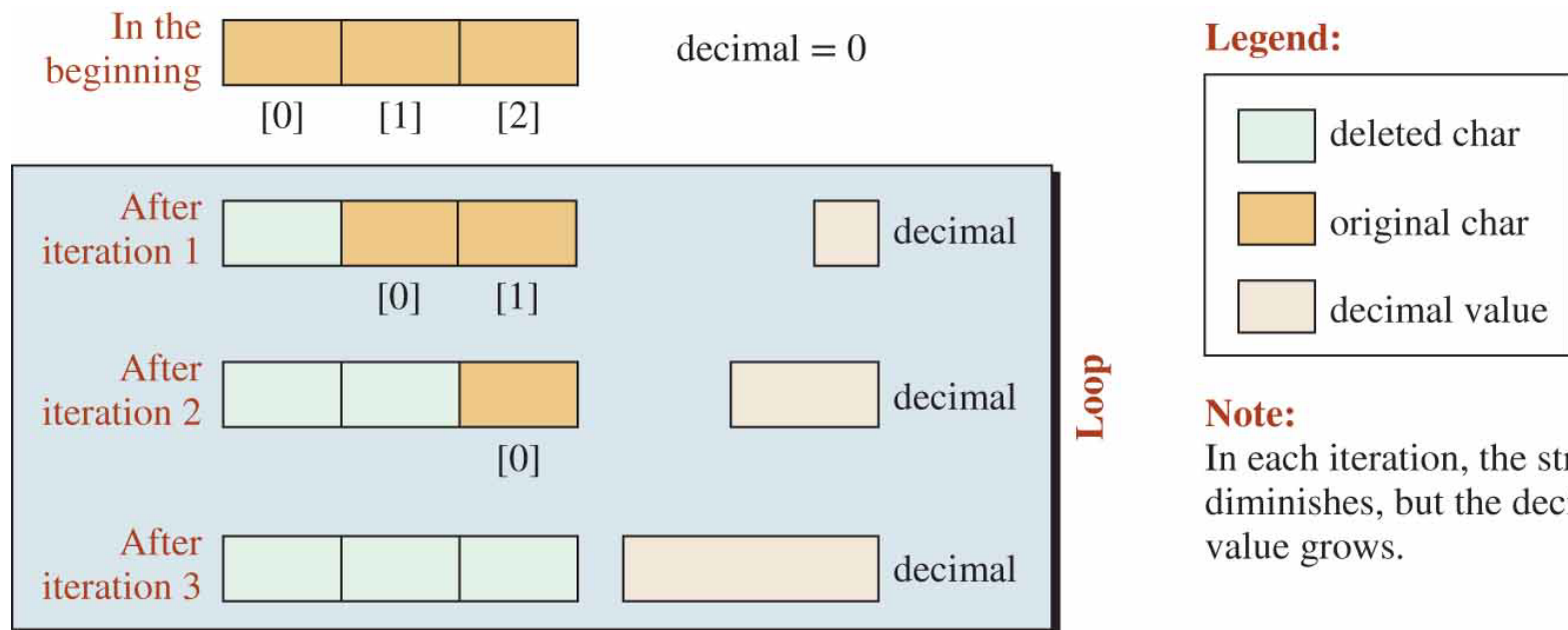
System	Base	Symbols
binary	2	0, 1
octal	8	0, 1, 2, 3, 4, 5, 6, 7
decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Conversion in Positional Number System

Conversion from Any Other Base to Decimal

Figure 10.14 shows how we convert a string of three characters to an integer.

Figure 10.14 *Converting from any base to decimal*



Conversion in Positional Number System

**We use a loop that is controlled by the size of the string.
When the string is empty, we exit the loop.**

Before starting the loop, we set the value of the decimal variable to 0.

In each iteration, we multiply the previous value of decimal by its base.

Table 10.4 *Algorithm to convert a string to a decimal*

```
set base                                // set base to 2, 8, 16
decimal = 0
input string
while (string not empty)
{
    decimal *= base;
    ch = popFront (string)              // Use popFront function developed before
    decimal += findValue (ch)           // Use a function to change ch to its value
}
output decimal
```

Conversion in Positional Number System

Program 10.27 *Changing a binary string to a decimal integer*

```
1  /*****
2   * A program to change a binary string to a decimal integer.   *
3   *****/
4  #include "customized.h"
5  #include <string>
6  #include <iostream>
7  using namespace std;
8
9  /*****
10   * A function to change a numeric character to its equivalent *
11   * integer value.                                           *
12   *****/
13  int findValue (char ch)
14  {
15      return static_cast <int>(ch) - 48; // ASCII code: '0'
16  }
17  int main ( )
18  {
19      // Declaration, inputting, and validation of binary string
20      string binary;
```

Conversion in Positional Number System

Program 10.27 Changing a binary string to a decimal integer

```
21  do
22  {
23      cout << "Enter binary string: ";
24      getline (cin, binary);
25  } while (binary.find_first_not_of ("01") < binary.size());
26  // Initialization and calculation of decimal integer
27  int base = 2;
28  int decimal = 0;
29  while (!binary.empty())
30  {
31      decimal *= base;
32      char ch = popFront (binary);
33      decimal += findValue (ch);
34  }
35  cout << "Decimal value: " << decimal;
36  return 0;
37 }
```

Conversion in Positional Number System

Program 9.21 *Changing a binary string to a decimal integer*

Run:

Enter binary string: 11 101

Enter binary string: 11101

Decimal value: 29

Run:

Enter binary string: 1181

Enter binary string: 11111

Decimal value: 31

Run:

Enter binary string: 111000111

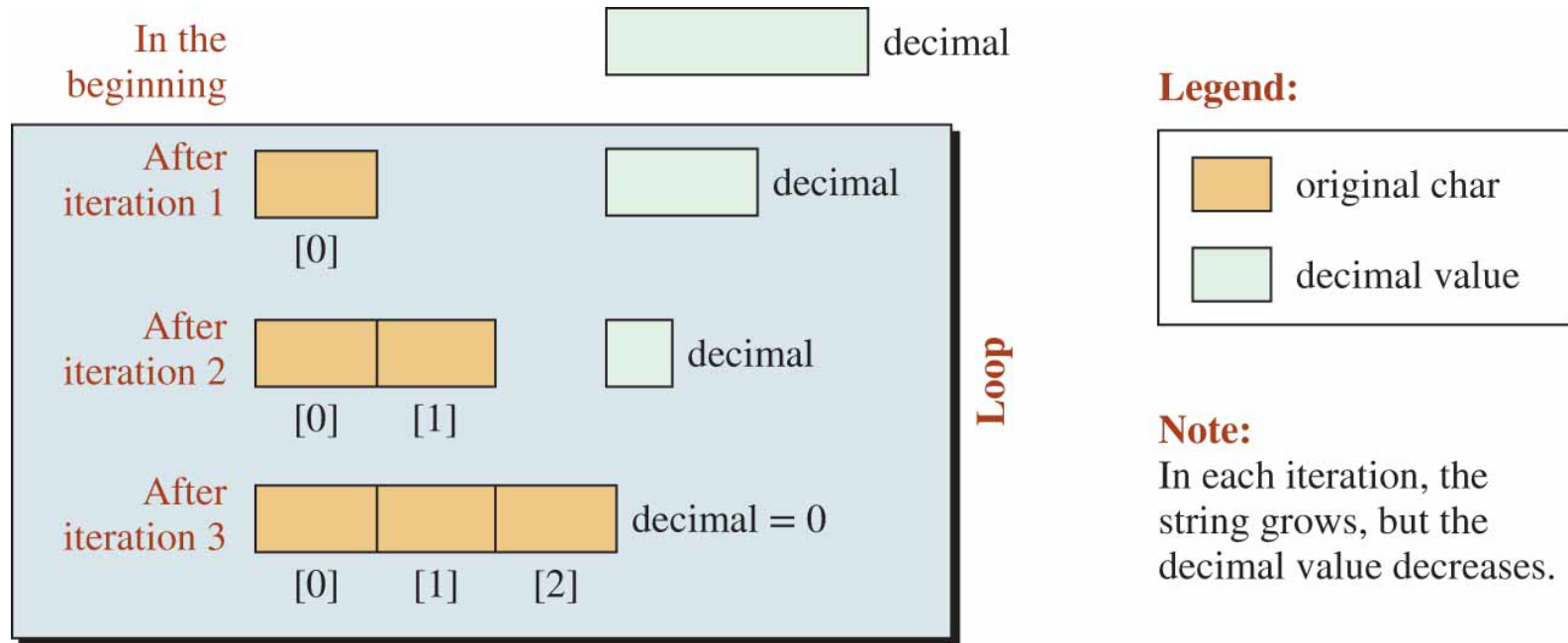
Decimal value: 455

Conversion in Positional Number System

Conversion from Decimal to Any Other Base

Figure 10.15 shows how we convert a decimal to a string of three characters.

Figure 10.15 *Converting from decimal to other bases*



Conversion in Positional Number System

We use a loop that is controlled by the value of the decimal number.

When the value is 0, we stop.

Before starting the loop, the string is empty.

In each iteration, we divide the previous value of decimal by base and get the remainder.

We then change the resulting value to a character and push it to the front of the string.

Conversion in Positional Number System

Table 10.5 *Algorithm to convert a decimal to a binary string*

```
set base                                // set base to 2, 8, 16
input decimal
while (decimal > 0)
{
    value = decimal % base;
    ch = findChar (value)                // Use a function to change value to char
    string.pushFront (ch)                // Use pushFront function developed before
}
output string
```

Conversion in Positional Number System

Program 10.28 Converting from decimal to binary

```
1  /*****
2   * A program to change a decimal numeral to a binary string      *
3   *****/
4  #include <string>
5  #include "customized.h"
6  #include <iostream>
7  using namespace std;
8
9  /*****
10   * A function to change an integer to a character using the      *
11   * char function.                                                *
12   *****/
13 char findChar (int digit)
14 {
15     return char (digit + '0');
16 }
17
18 int main ( )
19 {
20     // Declaration of variables
```


Conversion in Positional Number System

Program 10.28 Converting from decimal to binary

```
21  int decimal;
22  int base = 2;
23  string strg;
24  // Input and validation of decimal number
25  do
26  {
27      cout << "Enter a positive decimal: " ;
28      cin >> decimal;
29  } while (decimal <= 0);
30  // Conversion to binary
31  while (decimal > 0)
32  {
33      int digit = decimal % base;
34      char ch = findChar (digit);
35      pushFront (strg, ch);
36      decimal /= base;
37  }
38  // Outputting binary
39  cout << "Binary: " << strg;
40  return 0;
```

Conversion in Positional Number System

Program 10.28 *Converting from decimal to binary*

```
41 }
```

Run:

Enter a positive decimal: 35

Binary: 100011

Run:

Enter a positive decimal: 7

Binary: 111

Run:

Enter a positive decimal: 126

Binary: 1111110

What's Next?

Reading Assignment

- ☐ Read Chap. 11. Relationships among Classes

Thank you

E-mail: youngcha@konkuk.ac.kr

