Strings & Characters

CS 150 – C++ Programming I
Lecture 5

Characters

- The `char` type is a built-in C++ type
  - Unlike Java, the C++ `char` is a 7-bit ASCII character set
    - Means there are only `128` characters defined
  - For larger character sets, C++ has `wchar_t`
  - Since C++11 `char16_t` and `char32_t` added for Unicode
  - For arithmetic use `signed char` or `unsigned char`
- Header `<cctype>` has functions (macros) used for `char`
  - Headers that start with `c` are "inherited" from the C library
  - Classification: `isdigit(c)`, `isalpha(c)`, `isspace(c)`, etc.
  - Conversion: `tolower(c)`, `toupper(c)`
    - Only converts up appropriate

C++ and Strings

- A sequence of characters (text) is called a "string"
  - C++ has two kinds of strings
  - The built-in string, inherited from C
    - Arrays of `char` which are terminated with a `0` byte
    - String literals (`"hello world"`) are C-style strings
    - You'll study these when we look at pointers and arrays
- The C++ standard library type called `string`
  - A class type, not built in to the C++ language
  - You must `#include <string>`
  - A C++ string literal (C++14+) can be written as `"hello"s`

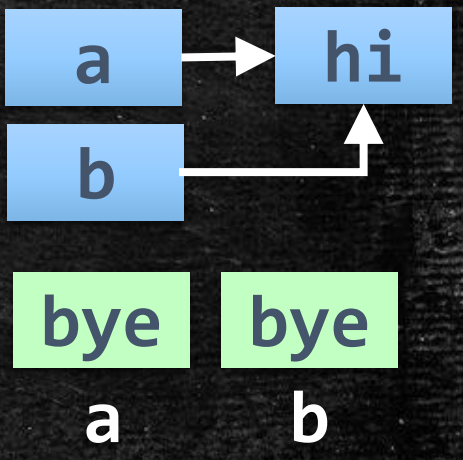
Creating String Variables

- A class type so you create them with a constructor
  - Syntax is different from Java; you don't use new
    - `string s = new string("Hello World");`
- You don't need to repeat the type name in C++

```
- string s;                          // empty string ""
- string s = "Hello World";
- string s{"Hello World"};           // or parentheses
- string dashes(50, '-');            // 50 dashes
- string ell("hello", 1, 3); // ell
- string raw = R"("\hello\")"; // "\hello\"
```

C++ string vs. Java String

- The C++ `string` type is a library value type
  - C++ library types act like the built-in types
- Different than Java's reference types
  - `String a = "hi", b = a; // 1 String object`
- C++ uses value (or copy) assignment (unlike Java)
  - `string a = "bye", b = a; // 2 string objects`
- Support native operator comparison (unlike Java)
  - `string a = "zebra", b = "aardvark";`
  - `if (a < b)` … or `if (a == b)`



C++ string Operations

- Strings may be modified (Java Strings are immutable)
  - `string str{"hello"}; str[0] = 'J'; // now Jello`
- Strings may be concatenated to literals & char
  - `string str1 = "hello", str2 = "world";`
  - `string str3 = str1 + " " + str2 + '!';`
- Cannot concatenate C-string literals or numbers (like Java)
  - `string str1 = "hello" + " " + "world!"; // illegal`
  - `string str2 = "Amount: " + 23.45;  // illegal`

String Input & Output

- Use `cin` & `cout` for input and output
  - `string first_name;`
  - `cout << "First name: "; //prompt (end in space)`
  - `cin >> first_name;`
    - Reads one token. Stops on any whitespace character
- Read a line with the `getline()` function in `<string>`
  - `string full_name;`
  - `cout << "Full name (last, first): ";`
  - `getline(cin, full_name);`

String Member Functions

- As in Java, objects have methods associated with them
  - In C++, these are called member functions
- The number of characters: `str.size()`
  - May also use `length()`, but `size()` is shorter
  - Returns an unsigned number: `string::size_type`
  - Reduce typing by using general type `size_t`
  - Don't use `int` or compare to `int` variable

```
for (int i = 0; i < s.size(); i++)
                  ~~^~~~~~~~~~
```

Selecting Characters

- Select individual characters by subscripting (index)
  - Legal subscripts are from `0` to `str.size() - 1`
- Subscript operator (`[]`) is not ranged checked

```
string str = "hello";
str[0] = 'j'; // OK
str[5] = '!'; //Undefined behavior
```

- `at()` member function is ranged checked and preferred

```
string str = "hello";
str.at(0) = 'j'; // OK
str.at(5) = '!'; //Safe!!! Throws exception
```

Extracting Substrings

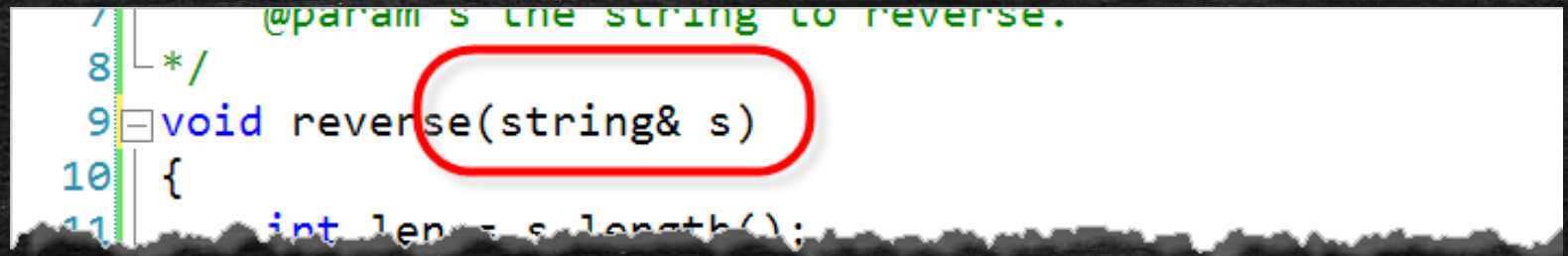
- You can copy a portion of a string into another string
  - In Java do this with the method named substring()
  - Uses two indexes, pos1 and pos2
  - ```
    String str = "Hello World!;"
    String subs = str.substring(6, 11); // World
    ```
- In C++ we use a member function named substr()
  - ```
    string str = "Hello World!";
    string sub1 = str.substr(6, 5); // grab 5
    ```
  - Out of range if pos1 is > str.size()
- Second version: substr(pos1)

References

- A C++ reference is a derived type
  - Acts as an alias for a variable (not a new variable)
  - `int a = 42; int& b = a; // one variable, two names`
  - NOT like references in Java/C#
- References can only refer to Lvalues (aka variables)
  - `int& n = 42; // ERROR – 42 is not a variable`
- No conversions - only exact type matches
  - `double pi = 3.14159;`
  - `int& pi2 = pi;  // ERROR – pi is not an int`
  - `double& pi3 = pi;  // OK – exact match`

Reference Parameters

- A reference parameter is also **not** a new variable
  - Indicate by & between parameter's type and name



- Changing a reference parameter changes the argument
  - These are called output parameters

```
string str = "hello";
reverse(str);              // named s inside function
cout << str << endl;       // olleh
```

Constant References

- Reference parameters are much more efficient than value parameters for large objects, like `string`
  - Because no copies are made, less memory is used
  - Because no constructors are called, they run faster
- But, they are also more "dangerous"
  - May accidentally change a variable you didn't mean to
  - C++ prevents this by providing constant references
  - `void print(const string& s) …`
- Exercise: Using the `substr()` member function

Searching for Strings & Characters

- Search a string using `str.find(arg)`
  - Returns position where *arg* is found in *str*
  - *arg* can be `string`, `char` or c-string literal
  - Returns `string::npos` (no-position) if not found
- Optional second argument: the starting position
  - `str.find(arg, pos);`
  - Searches for *arg* starting at *pos* in the string *str*.
- Search from the right with `str.rfind(arg)`
- Exercise: `Initials`

Range or Iterator Loops

- C++11 introduced a new, simplified range-based *for* loop
  - Includes *string*, *vector* and the built-in array types
  - *for (type e : collection) . . .*
- Three variations
  - *for (auto e : col)* *// value*
  - *for (auto& e: col)* *// reference*
  - *for (const auto& e: col)* *// const ref*
- Exercises: `Flipper`, `Upper`