CS2311 Computer Programming

LT07: String

Computer Science, City University of Hong Kong

Semester B 2022-23

Today's Outline

- *char* recap
- C string basics
- Reading and printing C strings
- Common string functions
- Safety of string functions

Recap: char

char is a data type that represents a single character or "glyph"

```
char letterA     = 'A';
char plus        = '+';
char zero        = '0';
char space       = ' ';
char newLine     = '\n';
char tab         = '\t';
char singleQuote = '\'';
char backSlash   = '\\';
```

- In C++ language, a char type is represented by an integer
- Therefore, a character can also be treated as an integer
- Examples:

```
cout << 'a';              // a
cout << (int)'a';         // 97
cout << 'a' + 1;          // 98
cout << (char)('a' + 1); // b
```

char: Example

- Write a program which reads a character from the user and output the character type
- The program should distinguish between the following types of characters
  - An upper-case character ('A'-'Z')
  - A lower-case character ('a'-'z')
  - A digit ('0'-'9')
  - Special character (e.g., '#', '$', etc.)

```
#include <iostream>
using namespace std;
int main() {
  char c;
  cin >> c;

  if ('A'<=c && c<='Z') // 'A'-'Z'
    cout << "An upper-case character\n";
  else if ('a'<=c && c<='z') // 'a'-'z'
    cout << "A lower-case character\n";
  else if ('0'<=c && c<='9') // '0'-'9'
    cout << "A digit\n";
  else
    cout << "Special character\n";

  return 0;
}
```

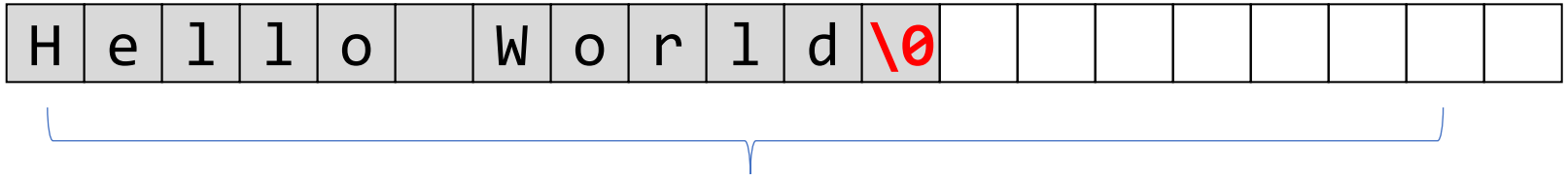
cstring vs std::string

- In C++, there are two types of strings
  - cstring: inherited from the C language
    - #include <cstring>
  - string: *class* defined in std library
    - #include <string>
    - Class and object (introduced in later lecture)

C String

- A C string is a char array terminated by '\0'
- '\0': null character representing the end-of-string sentinel
- Consider the definition and initialization of char str[20]

```
char str[20] = "Hello World"; // '\0' will be added automatically
```



str may store a string with maximum of 19 characters

C String: '\0'

- The null character, i.e., '\0', is used to mark the end of a C string

- '\0' is a single character (although written in two symbols)

- It's used to distinguish a C string from an ordinary array of characters
  - a C string must contain a null character

C String: Declaration and Initialization

- Declare a C string with one more character than needed
  - reserve one slot for '\0'
- A string can be declared in two ways
  - With initialization: char identifier[] = string constant / string literal;

e.g., `char studentID[] = "51234567";`

`char HKID[] = "a123456(7)";`

  - Without initialization: char identifier[required_size+1];

e.g., `char studentID[8+1];`

`char HKID[10+1];`

C String: Declaration and Initialization

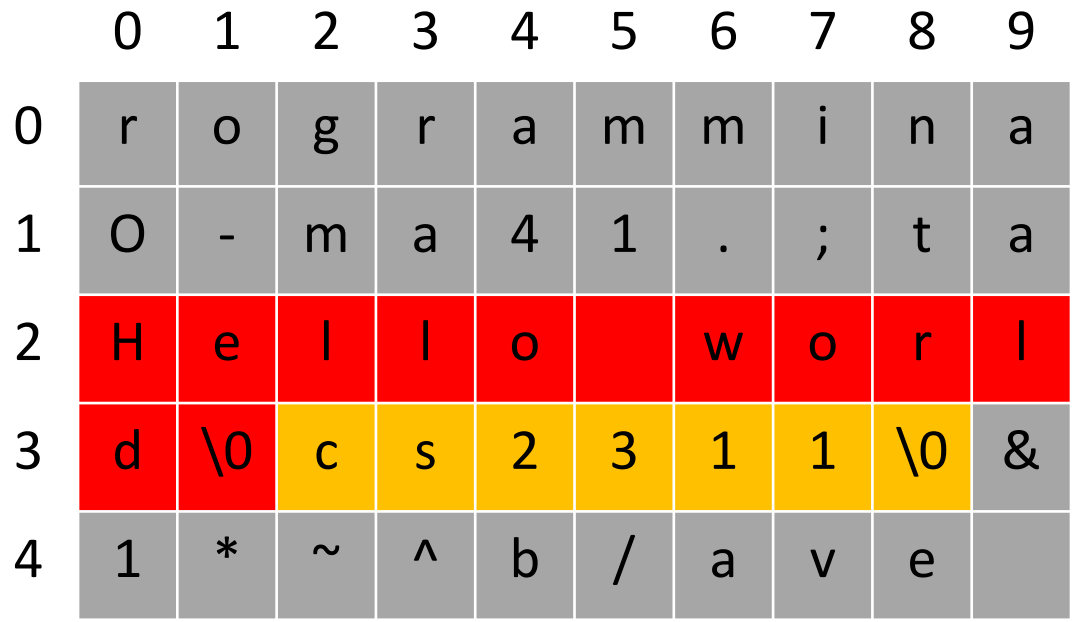
- However, you cannot initialize a string after declaration
  - `char name[10];`
  - `name = "john";` `// error C2440: '0': cannot covert from 'const char[5]' 'to char[10]'`

- Note the difference between char and string
  - `char grade   = 'A';` `// a character`
  - `char grade[] = "A";` `// a C string terminated with '\0'`
  - `char grade   = "A";` `// error C2440: '=': cannot convert from 'const char[2]' to 'char'`

C String: Storage

- A C string is stored in main memory continuously
- the C strin variable stores the starting memory address of the string content

```
char s1[]="Hello World"; // s1=20
char s2[]="cs2311";      // s2=32
```



Passing String to Functions

- Example
  - Write a function to count the frequency of a character (e.g., 'a') in a string
- Functions
  - `count`: given a character and a string as input, return the frequency of the character in the string
  - `main` function: call `count` function

```
int count(char s[100], char c) {
  int frequency=0;
  int i=0;
  while (s[i]!='\0') {
        if (s[i]==c)
              frequency++;
        i++;
  }
  return frequency;
}
```

Today's Outline

- *char* recap
- C string basics
- Reading and printing C strings
- Common string functions
- Safety of string functions

Reading and Printing C Strings

```
#include <iostream>
using namespace std;
int main() {
   char word[5];
   cin >> word;  // read a string
   cout << word; // print a string
   return 0;
}
```

Printing C Strings

- Recall: a C string is stored in main memory continuously
- Recall: the C string variable stores the starting memory address of the string content
- When a C string, say **str**, is passed to an output function (e.g., cout), the function will print all memory content starting from the address specified by **str**, *until a '\0' is encountered*

- What will be printed?

```
int main() {
   char s1[] = "abc";
   char s2[] = "def";
   s1[3] = '+';
   cout << s1 << endl << s2 << endl;
   return 0;
}
// abc+def
// def
```

Reading C Strings

- `cin >> str` will terminate when a whitespace character is encountered
  - whitespace: space, tab, newline …

```
char s1[20], s2[10];
cin  >> s1;   // user input "hello world\n"
              // cin reads "hello" and stops when ' ' is encountered;
              // s1 gets "hello", '\0' is automatically added
              // "world\n" is stored in buffer to be consumed later
cin  >> s2;   // since there's content left in buffer, cin will read buffer first
              // i.e., no user input is needed
              // cin reads "world" in buffer and stops when '\n' is encountered
              // s2 gets "world", '\0' is automatically added
cout << s1;   // will print "hello"
cout << s2;   // will print "world"
```

Reading C Strings

- `cin >> str` will terminate when a whitespace character is encountered
  - whitespace: space, tab, newline …

```
char s1[20], s2[10];
cin  >> s1;   // user input "hello world\n"
              // cin reads "hello" and stops when ' ' is encountered;
              // s1 gets "hello", '\0' is automatically added
              // "world\n" is stored in buffer to be consumed later
cin  >> s2;   // since there's content left in buffer, cin will read buffer first
              // i.e., no user input is needed
              // cin reads "world" in buffer and stops when '\n' is encountered
              // s2 gets "world", '\0' is automatically added
cout << s1;   // will print "hello"
cout << s2;   // will print "world"
```

Reading C Strings

- `cin >> str` will terminate when a whitespace character is encountered
  - whitespace: space, tab, newline …

```
char s1[20], s2[10];
cin  >> s1;   // user input "hello world\n"
              // cin reads "hello" and stops when ' ' is encountered;
              // s1 gets "hello", '\0' is automatically added
              // "world\n" is stored in buffer to be consumed later
cin  >> s2;   // since there's content left in buffer, cin will read buffer first
              // i.e., no user input is needed
              // cin reads "world" in buffer and stops when '\n' is encountered
              // s2 gets "world", '\0' is automatically added
cout << s1;   // will print "hello"
cout << s2;   // will print "world"
```

Reading a Line: get() Loop

- **cin >> str** stops when a whitespace is encountered
  - How to get a line of chars from user input?
- **get():** member function of cin to read in one character from input
  - **>>** skipping over whitespace but **get()** won't
- syntax: char c;
  cin.get(c);

```
#include <iostream>
using namespace std;
// read user input,        until
// the end of line (i.e., '\n') is reached


int main() {



  char c;
  do {
     cin.get(c);
     cout << c;

  } while (c!='\n);
  return 0;
}
```

Reading a Line: get() Loop

- **cin >> str** stops when a whitespace is encountered
  - How to get a line of chars from user input?
- **get():** member function of cin to read in one character from input
  - **>>** skipping over whitespace but **get()** won't
- syntax: `char c;`
  `cin.get(c);`

```
#include <iostream>
using namespace std;
// read user input to str, until
// the end of line (i.e., '\n') is reached
// or str is full
int main() {
  char str[20];
  int i = 0;
  char c;
  do {
     cin.get(c);
     cout << c;
     str[i++] = c;
  } while (c!='\n' && i<20);
  return 0;
}
```

Reading a Line: getline

- **getline():** predefined member function of cin to read a line of text (including space)

- Two arguments:
  - a C string variable to receive the input
  - size of the C string

```
#include <iostream>
using namespace std;
int main() {
  char s[20];
  while (true) {
    cin.getline(s, 20);
    cout << "\"" << s << "\"" << "\n";
  }
  return 0;
}
```

Reading a Line: getline

- What if
  - Input is longer than the string variable?
  - End of the source characters is reached?
  - Error occurred?

- Internal state flags (eofbit, failbit, badbit) of cin object will be set

- To reset those flags, call method **clear()** of cin, e.g., **cin.clear();**

Example

- Input "12345" and see what will be printed

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
  char s[5];
  int i = 0;
  while (true) {
    cin.getline(s, 5);
    cout << i++ << ": " << s << endl;
  }
  return 0;
}
```

Example

- Input "12345" and see what will be printed

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
  char s[5];
  int i = 0;
  while (true) {
    cin.getline(s, 5);
    cout << i++ << ": " << s << endl;
  }
  return 0;
}
```

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
  char s[5];
  int i = 0;
  while (true) {
    cin.getline(s, 5);
    cin.clear(); // clear state flag so cin can
                 // continue
    cout << i++ << ": " << s << endl;
  }
  return 0;
}
```

Today's Outline

- *char* recap
- C string basics
- Reading and printing C strings
- Common string functions
- Safety of string functions

strlen

- **strlen(str)**: returns the number of chars (before '\0') in C string **str**
  - '\0' does NOT count towards the length
- In comparison, recall that sizeof returns array size (number of bytes)

```
char myStr[20] = "Hello World!";
int len = strlen(myStr);
int siz = sizeof(myStr);
cout << len << "\n"; // 12
cout << siz << "\n"; // 20
```

strlen

- Example: write a program to print the shortest string in a string array

```
#include <iostream>
#include <cstring>
using namespace std;
#define MAX_LEN 100
int main() {
  char s[5][MAX_LEN] = {
    "Hi World", "Hi", "cs2311",
    "Hello", "Hello World"
  };
  cout << s[shortest(s, 5)];
  return 0;
}
```

```
int shortest(char s[][MAX_LEN], int n) {
  int i, j=0, min_len=strlen(s[0]);
  for (i=1; i<n; i++) {
    int len_i = strlen(s[i]);
    if (len_i < min_len) {
      min_len = len_i;
      j = i;
    }
  }
  return j;
}
```

strlen

- Caution: strlen scans the entire string when invoked

```
#define N 1000000

int main() {
   char s[N];
   for (int i = 0; i < N-1; i++)
      s[i] = char('a' + rand()%26);
   s[i] = '\0';

   int frequency = 0;
   for (int i = 0; i < strlen(s); i++) {
      if (s[i] == 'd')
         frequency++;
   }
   cout << frequency << "\n";
   return 0;
}
```

strlen

- Caution: strlen scans the entire string when invoked

```
#define N 1000000

int main() {
   char s[N];
   for (int i = 0; i < N-1; i++)
      s[i] = char('a' + rand()%26);
   s[i] = '\0';

   int frequency = 0;
   for (int i = 0; i < strlen(s); i++) {
      if (s[i] == 'd')
         frequency++;
   }
   cout << frequency << "\n";
   return 0;
}
```

```
#define N 1000000

int main() {
   char s[N];
   for (int i = 0; i < N-1; i++)
      s[i] = char('a' + rand()%26);
   s[i] = '\0';

   int frequency = 0, len = strlen(s);
   for (int i = 0; i < strlen(s)len; i++) {
      if (s[i] == 'd')
         frequency++;
   }
   cout << frequency << "\n";
   return 0;
}
```

strlen: Implement by Yourself

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

int main() {
   char s[20]="Hello world";
   int len = 0;
   while (s[len] != '\0')
      len++;
   cout << len << endl;
   return 0;
}
```

- The implementation in cstring uses pointer
- Same for other cstring functions introduced in this lecture
- Pointer will be introduced in later lecture

strcpy

- **strcpy(dst, src)**: copies the characters of string **src** into string **dst**, stops when '\0' is encountered in **src**

```
char s1[6];
strcpy(s1, "hello");
char s2[6];
strcpy(s2, s1);
s2[0] = 'c';
cout << s1 << endl; // hello
cout << s2 << endl; // cello
```

strcpy: Implement by Yourself

```
#include <iostream>
using namespace std;
int main() {
   char src[]="Hello world";
   char dst[15];
   int i;
   for (i=0; src[i]!='\0'; i++)
      dst[i] = src[i];
   dst[i] = '\0';
   cout << dst;
   return 0;
}
```

1. Use a loop to read characters one by one from **src** until a '\0' is read
2. copy the character to the corresponding position of **dst**
3. put a '\0' at the end of **dst**

- The following expression doesn't copy string content

```
dst = src;
```

- The following expression doesn't compare string contents

```
if (s1==s2)
```

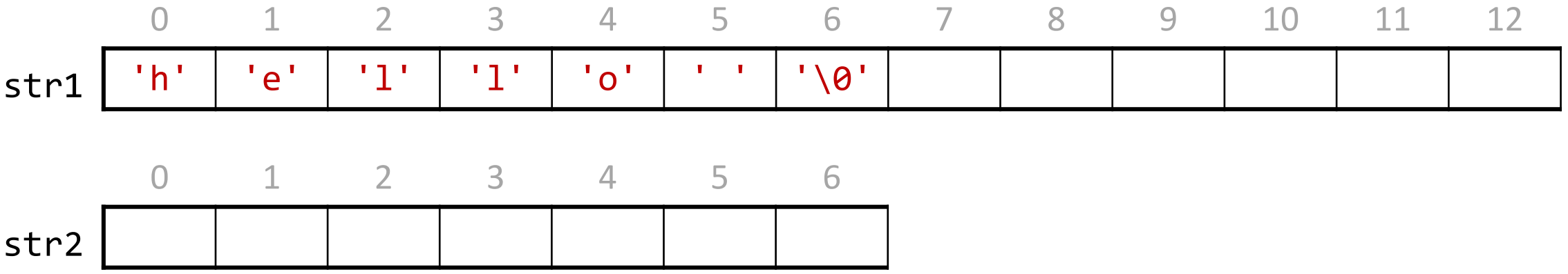
strcat

- We **cannot** concatenate C strings using +: this adds addresses!
- Instead, use strcat
  - **strcat(dst, src)** concatenates the contents of **src** into **dst,** i.e., copies the characters in **src** to the end of **dst**, until '\0' is encountered in **src**

```
char str1[13];
strcpy(str1, "hello ");
strcat(str1, "world!"); // removes old '\0', adds new '0' at the end
cout << str1;
```

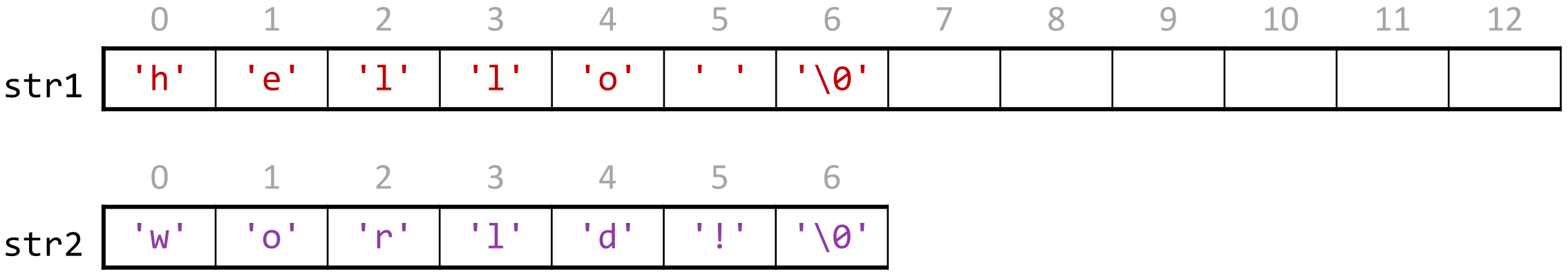
strcat

```
char str1[13];
strcpy(str1, "hello ");
char str2[7];
strcpy(str2, "world!");
strcat(str1, str2);
```



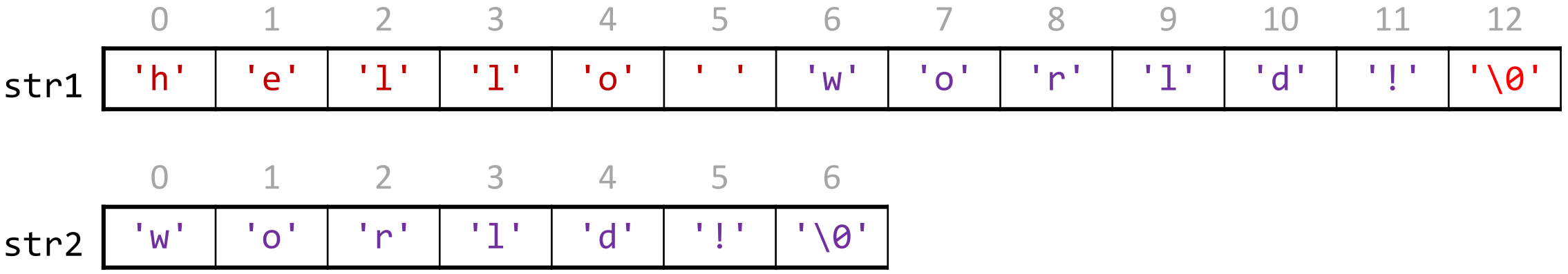
strcat

```
char str1[13];
strcpy(str1, "hello ");
char str2[7];
strcpy(str2, "world!");
strcat(str1, str2);
```



strcat

```
char str1[13];
strcpy(str1, "hello ");
char str2[7];
strcpy(str2, "world!");
strcat(str1, str2);
```



strcat: Implement by Yourself

```
int main() {
   char s1[20] = "Welcome to ";
   char s2[20] = "cs2311";
   long s1_len = strlen(s1);
   long s2_len = strlen(s2);
   char s[100];
   for (int i = 0; i < s1_len; i++)
      s[i] = s1[i];
   for (int i = s1_len; i < s1_len+s2_len; i++)
      s[i] = s2[i-s1_len];
   s[s1_len + s2_len] = '\0';
   cout << s << endl;
   return 0;
}
```

strcmp

strcmp(str1, str2) compare **str1** and **str2,** until

- encounters a pair of characters that don't match
- reaches the end of str1 or str2 (i.e., encounters '\0' in str1 or str2)

- Let **c1** and **c2** be the pair of characters in **str1** and **str2** that don't match
  - **< 0**: if **c1 < c2** (i.e., **str1** is smaller than **str2** in alphabet)
  - **> 0**: if **c1 > c2** (i.e., **str1** is greater than **str2** in alphabet)
  - **return 0** if **str1** and **str2** are identical

strcmp

strcmp(str1, str2) compare **str1** and **str2,** until

- encounters a pair of characters that don't match
- reaches the end of str1 or str2 (i.e., encounters '\0' in str1 or str2)

- Let **c1** and **c2** be the pair of characters in **str1** and **str2** that don't match
  - **< 0**: if **c1 < c2** (i.e., **str1** is smaller than **str2** in alphabet)
  - **> 0**: if **c1 > c2** (i.e., **str1** is greater than **str2** in alphabet)
  - **return 0** if **str1** and **str2** are identical

- e.g.,
```
cout << strcmp("abc",  "abc") << "\n"; //  0
cout << strcmp("abc", "abcd") << "\n"; // -1
cout << strcmp("abcd", "abc") << "\n"; //  1
cout << strcmp("abc",  "abd") << "\n"; // -1
```

strcmp: Implement by Yourself

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

#define MAX_LEN 20

int main() {
    char s1[MAX_LEN] = "abcdef";
    char s2[MAX_LEN] = "abcdEF";
    cout << compare(s1, s2) << endl;
    return 0;
}
```

```
int compare(char s1[MAX_LEN], char s2[MAX_LEN]) {
    int size = strlen(s1);
    for (int i = 0; i < size; i++) {
        if (s1[i] < s2[i]) {
            cout << "str1 is smaller than str2\n";
            return -1;
        } else if (s1[i] > s2[i]) {
            cout << "str2 is greater than str1\n";
            return 1;
        }
    }
    cout << "str1 is equal with str2\n";
    return 0;
}
```

Other String Functions

- **strncpy(dst, src, n)**
  - copies the first ***n*** characters of ***src*** to ***dst***.
  - if the end of ***src*** (signaled by '\0') is found before ***n*** characters have been copied, ***dst*** is padded with zeros until a total of ***n*** characters have been written to it
- **strncat(dst, src, n)**
  - appends the first ***n*** characters of ***src*** to ***dst***, plus a '\0'
  - if the length of ***src*** is less than ***n***, only the content up to '\0' is copied
- **strncmp(str1, str2, n)**
  - compares up to ***n*** characters of ***str1*** to those of ***str2***
  - it continues comparison until the characters differ, a '\0' is reached, or ***n*** characters match in both strings, whichever happens first

Other String Functions (cont'd)

- **strchr(str, ch) / strrchr(str, ch)**
  - character search: returns a pointer to the first occurrence of character ***ch*** in ***str*** or NULL if ***ch*** was not found in ***str***
  - ***strrchr*** finds the last occurrence
- **strstr(haystack, needle)**
  - string search: returns a pointer to the start of the first occurrence of C string ***needle*** in C string ***haystack***, or ***NULL*** if ***needle*** was not found in ***haystack***

Other String Functions (cont'd)

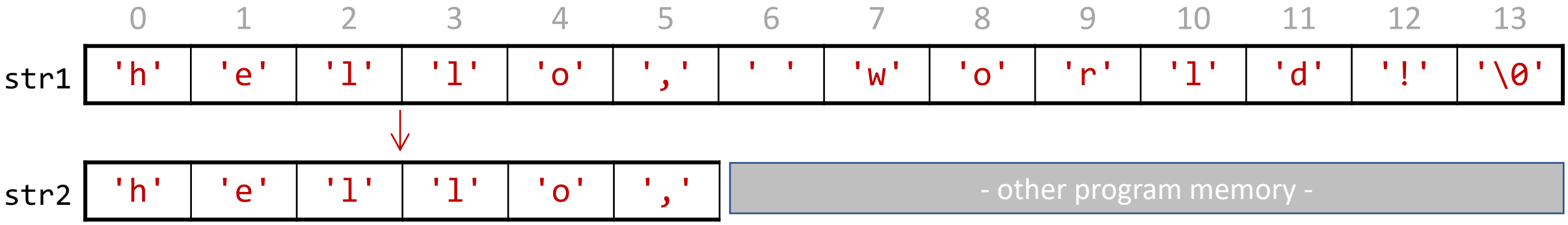
- **strspn(str, accept)**
  - returns the length of the initial part of ***str*** which contains only characters in ***accept***
- **strcspn(str, reject)**
  - returns the length of the initial part of ***str*** which does not contain any characters in ***reject***

```
char s1[] = "129th";
char s2[] = "ab123";
char digit[] = "1234567890";

int i = strspn(s1, digit);
cout << "The first " << i << "characters of s1";
cout << " are digits\n";

int j = strcspn(s2, digit);
cout << "The first " << j << "characters of s2";
cout << " are not digits\n";
```

Safety of String Functions

- **Recap:** strcpy(dst, src) copies characters in src to dst until '\0' is encountered in src
- What if src is longer than dst?

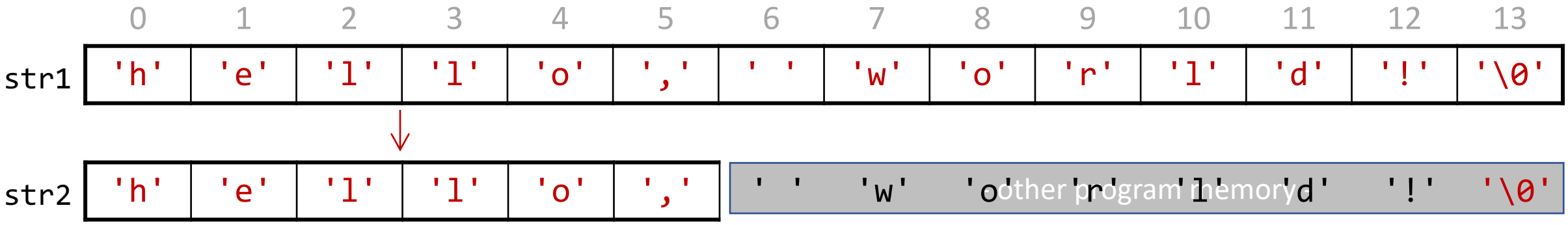
```
char str1[14];
strcpy(s, "hello, world!");
char str2[6];
strcpy(str2, str1);
```



Safety of String Functions

- **Recap:** strcpy(dst, src) copies characters in src to dst until '\0' is encountered in src
- What if src is longer than dst?

```
char str1[14];
strcpy(s, "hello, world!");
char str2[6];
strcpy(str2, str1);
```



Additional Notes

- strcpy and strcat are considered unsafe, as they don't check memory boundary
- In VS, the compiler refuses to run them by default
- You need to either
  - Add a pre-processor directive

    _CRT_SECURE_NO_WARNINGS
  - Use strcpy_s and strcat_s instead of strcpy and strcat

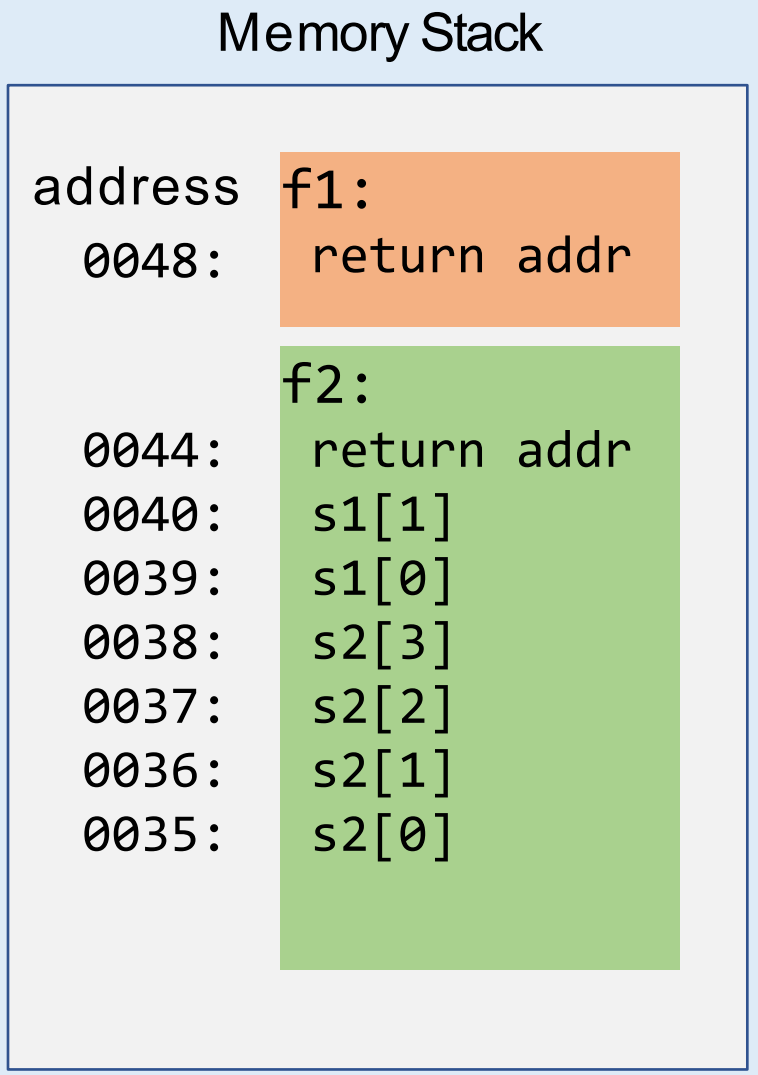
Buffer Overflow

- writing past memory bounds is called *buffer overflow*, which security vulnerabilities
- recall: each function has its own memory space

```
void f2() {
   char s1[2], s2[4];
   cin.getline(s2, 20);
   strcpy(s1, s2);
}
```

```
void f1() {
   f2();
}
```

- overwrite s1 in f2 allows an attacker to control return address． He can input binary malicious code in s2, and modify the return addr of f2 to run that code



L01: Introduction

- Von Neuman architecture

- Binary instruction <= Symbolic language <= High-level language

- External and internal view of computer program

L02: Data, Operators, and BasicIO

- Basic syntax
- Variable and constant
  - sizeof data types, implicit/explicit type conversion, char type and operations
- Operators
  - Efficient assignment operators, increment & decrement
- Basic IO
  - fixed, scientific, setprecision

L03: Control Flow - Conditional

- bool, type conversion from other types to bool
- Comparative operators: = vs ==
- Logic operators (&& and ||), short circuit, a<x<b vs a<x && x < b
- if: basic syntax, inline ternary, compound if
- switch: basic syntax, break, default

L04: Control Flow - Loop

- Basic loop structure
  - Initialization, loop condition, loop body, post loop statement
- while, do-while, for: basic syntax
- Nested loop
- break and continue

L05: Function

- Basic syntax of defining and calling a function
- Function prototype, header file
- Parameter passing
  - Parameter vs argument, pass-by-value, pass-by-reference
- Recursive functions
  - Basic case, break down (representation with a smaller version of the problem itself)
  - Iterative vs recursive

L06: Array

- Basic syntax for: definition and initialization
  - basic init, init without size, partial init, all zeros
- Read and write array
- Passing arrays to functions
- Operations: sizeof, compare, sort, sequential search
- Multi-dimensional array
  - define: int a[][3] = {1,2,3,4}; int a[2][] = {1,2,3,4};
  - storage: row major
  - passing to function