# CS2311 Computer Programming

## LT07: String

*Computer Science, City University of Hong Kong*

*Semester B 2022-23*

# About Midterm

- Week 8 lecture time (Mar 10 Friday)
- In classroom, on paper (written based)
  - Formal proof needed for request of absence
- **90 minutes**
- From Lec 1 (Intro) to Lec 6 (Array)
- 15% of final mark

# Quick Review: Array

- Array definition
- Array initialization
- Passing array to functions
- Array operations
- Multi-dimensional array

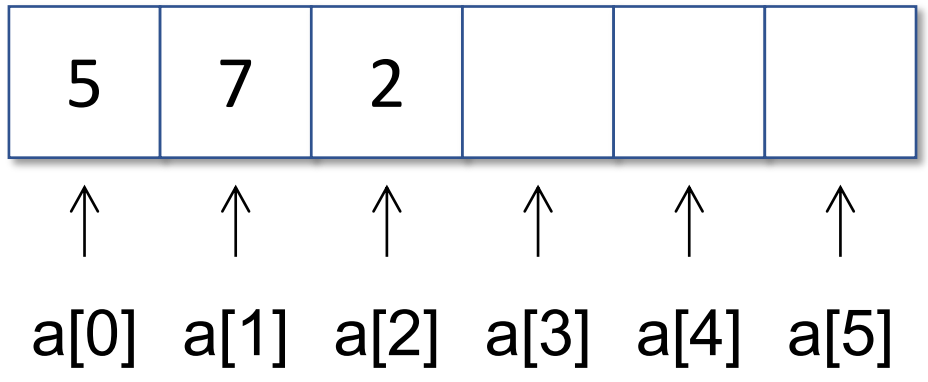
# Quick Review: What's an Array?

- Sequence of data items of the same type

  `data_type array_name[size]`

  - stored continuously
  - can be accessed by index, or subscript

```
int a[6];
a[0] = 5;
a[1] = 7;
a[2] = 2;
```



# Quick Review: Array Definition

- Set array size

```
const int n = 10;
int mark[n];

int mark[50*50];
```

```
int n = 10;
int mark[n];
```

```
int n; cin >> n;
int mark[n];
```

# Quick Review: Using #define to Set Array Size

- #define is a C++ predefined macro keyword

  Usage: `#define A B`

- which globally replaces all occurrences of A to B in the ENTIRE source code listing (all .cpp and all .h files)

- Examples:

```
#define N 100
#define SIZE 10
```

- Using #define to set array size

```
#define N 100
int mark[N];
```

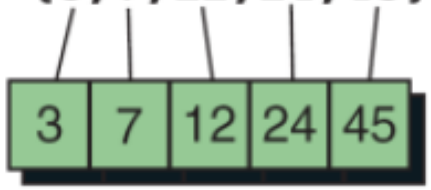
# Quick Review: Array Initialization

```
int a[3]={1,2,3};       // a has type int[4] and holds 1, 2, 3
int b[5]={1,2,3};       // b has type int[5] and holds 1, 2, 3, 0, 0
int c[4]={1};           // c has type int[4] and holds 1, 0, 0, 0
int d[3]={0};           // d has type int[3] and holds all zeros
int e[] ={1,2,3};       // e has type int[3] and holds 1, 2, 3

int f[2]={1,2,3};       // it's an error to provide more elements than array size
int g[];                // it's an error to declare an array without specifying size
```

# Quick Review: Array Initialization Summary

(a) Basic Initialization

```
int numbers[5] = {3,7,12,24,45};
```

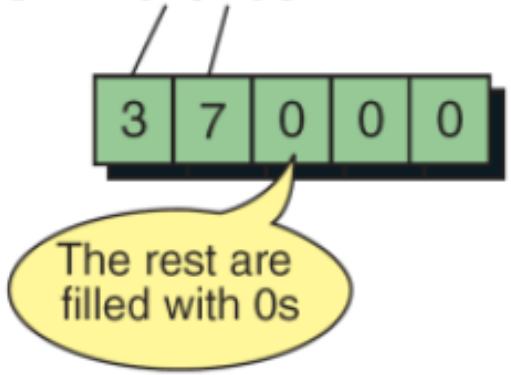



(b) Initialization without Size

```
int numbers[ ] = {3,7,12,24,45};
```

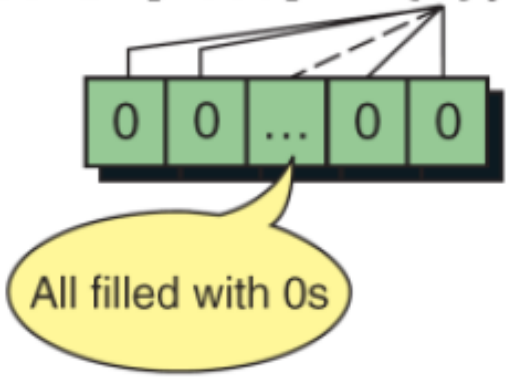

(c) Partial Initialization

```
int numbers[5] = {3,7};
```




(d) Initialization to All Zeros

```
int lotsOfNumbers [1000] = {0};
```



# Quick Review: Passing Arrays to Function

- To pass an array to a function, we only need to specify the array name
- Array is passed by pointer

the size of the array is optional, e.g., you can write `void f(int a[])`

```
void func(int a[3]){
  cout << a[0] << endl; // print 1
  a[0]=10;
}

void main () {
  int a[3]={1,2,5};
  func(a);
  cout << a[0] << endl; // print 10
}
```

if the content of a[i] is modified in func, the modification will persist even after the function returns (Call by pointer, more in later lectures)

only need to input the array name!

# Quick Review: Passing Arrays to Function

- The following program is invalid

```
void f(int x[20]) {
    …
}
void main() {
    int y[20];
    f(y[0]); // invalid, type mismatch
}
```

# Quick Review: sizeof

- Recall: sizeof(data_byte) gives the number of bytes of the data_type

```
cout << sizeof(int); // will print 4
```

- sizeof Array gives the total number of bytes occupied by that array

```
int a[4];
cout << sizeof(a); // will print 16
```

- How to calculate number of elements of an array?

# Quick Review: Compare Two Arrays

- We have two integer arrays, each with 5 elements

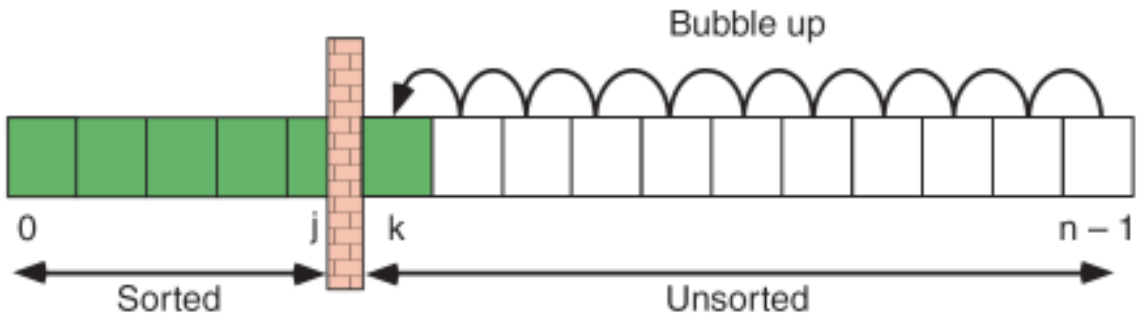
```
int array1[5] = {10, 5, 3, 5, 1};
int array2[5]; // will be entered by the user
```

- Compare whether array1 and array2 are equal
  - array equality: two arrays are equal if all of their elements are equal.
  - you have to compare all array elements one by one
  - the following code will generate wrong result

```
if (array1 == array2)
    cout << "the arrays are equal\n";
```

# Quick Review: Bubble Sort

- Repeat bubbling up for *n* rounds, where *n* is array size
- After round j (start from round 0), the array is divided into two parts:
- sorted (green): from a[0] to a[j]
- unsorted: from a[j+1] to a[n-1]




```
const int n = 6;
int a[n] = {23, 78, 45, 8, 32, 56};
int j, k, tmp;

for (j=0; j<n-1; j++) {     // outer loop
   for (k=n-1; k>j; k--) { // bubbling
      if (a[k]<a[k-1]) {
         tmp    = a[k];    // swap
         a[k]   = a[k-1];
         a[k-1] = tmp;
      }
   }
}

cout << "sorted: ";
for (j=0; j<n; j++)
   cout << a[j] << ' ';
cout << "\n";
```

# Quick Review: Searching

- Search sequentially

```
#define N 6
int sequentialSearch(int target, int a[N]) {
   for (int i=0; i<N; i++) {
      if (a[i] == x)
         return i;
   }
   return -1;
}
```

# Quick Review: Binary Search

- Assume the array is sorted (in ascending order)
- Compare the target with the middle element of the array
- If smaller, search in left
- If larger, search in right
- If equal, found

# Quick Review: Sort + Search

- Assume that you have a huge amount of data (out-of-order) and you need to frequently search in the database for different elements

- What should you do?

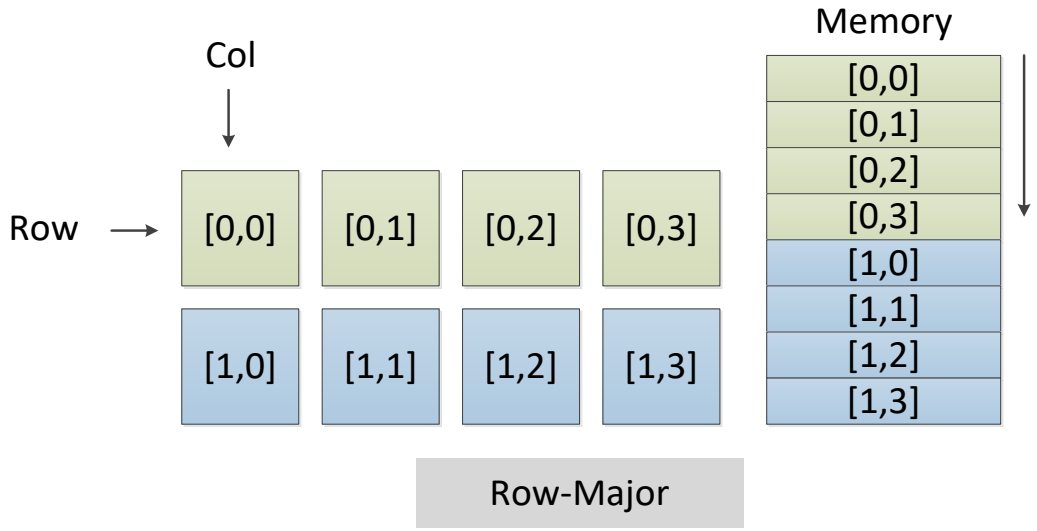
- What about if you only need to search once?

# Quick Review: Multi-dimensional Array

- Array with more than one index
  - on physical storage, multi-dimensional array is same as 1d array (*stored continuously in memory space*)
  - logical representation
- To define a 2d array, we specify the size of each dimension as follows

```
int page[2][3]; // [row][column]
```

- Stored in the "row-major" order



# Quick Review: 2D Array Initialization

- Assign initial values row by row

```
int page[2][3] = {{1,2,3},{4,5,6}};
```

- Assign initial values to the elements in the order they are arranged:

```
int page[2][3] = {1,2,3,4,5,6};
```

- Only assign initial values to some elements:

```
int page[2][3] = {{1},{4,5}};
```

```
1  0  0
4  5  0
```

- If all elements are assigned initial values, the length of the first dimension can be left unspecified:

```
int page[][3] = {1,2,3,4,5,6};
int page[2][] = {1,2,3,4,5,6}; X
```

# Quick Review: Passing 2D Array to Function

- The way to pass a 2D array is similar as the 1D array
- For example: define a function which reads a 2D array as the input and sort each row of the input 2D array

```
void sort2D(int x[][10]) {
    …
}
void main() {
    int y[20][10];
    sort2D(y);
}
```

the size of the first dimension is optional, while the size of the second dimension must be given

# Exercise 1

Which array definition(s) is generally correct?

a.
```
int x[1000] = { 1, 2, 3, 4};
```

b.
```
int x[] = { 1, 2, 3, 4 };
```

c.
```
const int SIZE = 1;
int x[SIZE];
```

d.
```
#define N 10000000
int mark[N];
```

e.
```
int x[3][] = {1,2,3,4,5,6};
```

f.
```
int x[3][2] = {{1},{4,5}};
```

# Exercise 2

```
int a[5] = {1, 2, 3, 4, 5};
func(a, 3);
cout << a[0] << " " << a[1] << " " << a[2] << endl;
func(a, 2);
cout << a[0] << " " << a[1] << " " << a[2] << endl;
```

```
void func(int a[3], int n){
      for (int i = 1; i < n; i++)
            a[i-1] = a[i]++;
}
```

What are the outputs?

# Exercise 3

What is the output produced by the following code?

```
0 0 0 0
0 1 1 1
0 2 3 3
0 3 5 6
```

```
int myArray[4][4] = {0}, index1, index2;
for (index1 = 1; index1 < 4; index1++)
       for (index2 = 1; index2 < 4; index2++)
              myArray[index1][index2] = myArray[index1-1][index2-1]+index1;

for (index1 = 0; index1 < 4; index1++) {
       for(index2 = 0; index2 < 4; index2++)
              cout << myArray[index1][index2] << " ";
       cout << endl;
}
```

# Initialization of Char Arrays

- `char str[3]={'a','b','c'};`

- `char str[3]="abc";` // str has type char[3] and holds 'a', 'b', 'c'

- `char str[] ="abc";` // str has type char[4] and holds 'a', 'b', 'c', '\0'

  // More details in the string lecture

# 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;
}
```

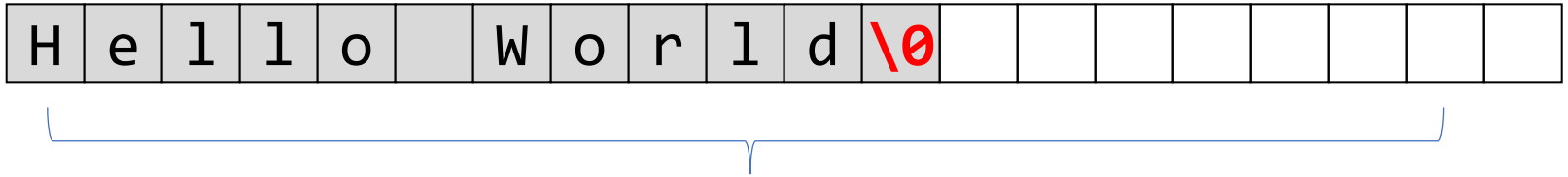
# cstring vs std::string

- 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[] = "a1234567";
      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

- 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[] = "a1234567";`

`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: Declaration and Initialization

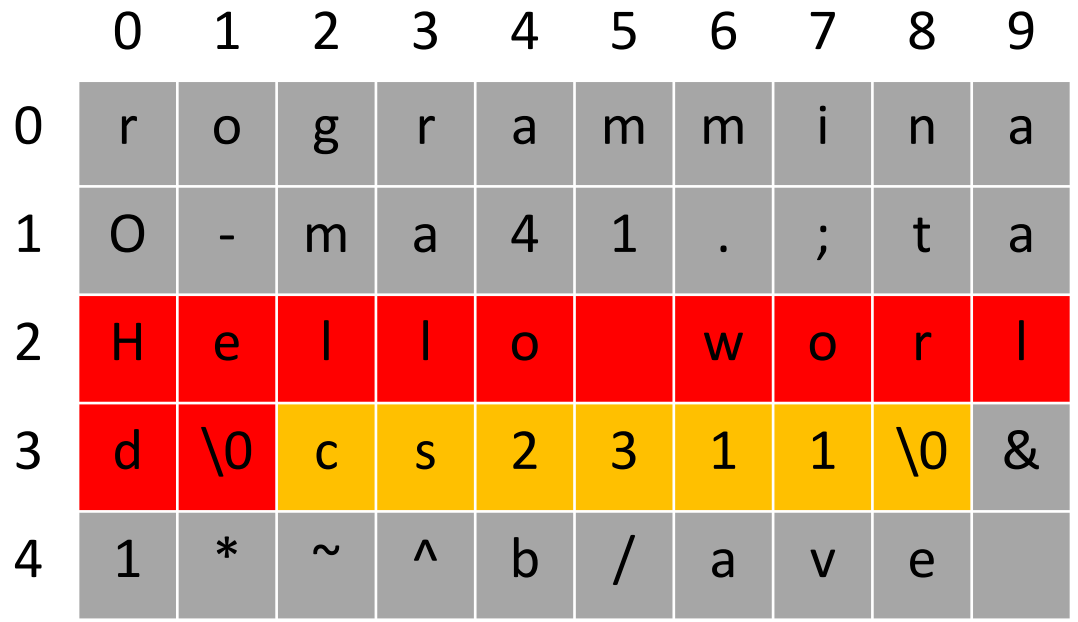
- However, you cannot initialize a string after declaration
  - `char name[10];`
  - `name = "john";` `// error C2440: '0': cannot covert from 'const char[5]' 'to char[10]'`

- 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 string 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

The size **100** is optional

```
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;
}

int main() {
  char T[100] = “hello world”;
  char t = ‘a’;
  cout << count(T, t) << endl;
  return 0;
}
```

# 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;
}
```

The array word can store 5 characters, but we can only use up to 4 character (the last character is reserved for null character).

# 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
```

# 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 …

- Suppose “Hello world” is the input

```
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 …

- Suppose “Hello world” is the input

```
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 …

- Suppose “Hello world” is the input

```
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 (before '\n' is encountered)?
- **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: get() Loop

- **cin >> str** stops when a whitespace is encountered
  - How to get a line of chars from user input (before '\n' is encountered)?
- **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;
}
```

```
0: 1234
1: 5
```

# Exercise 1

1. What is the length (maximum) of a string that can be placed in the string variable declared by the following declaration? Explain.

```
char s[6];
```

2. Which of the following statements are correct?

```
a. char str[10];  str = ”abc";
b. char shortString[] = "abc";
c. char shortString[2] = "abcd";
d. char shortString[10] = ‘a’;
```

# Exercise 2

Consider the following code
(and assume it is embedded in a complete and correct program and then run):

```
char a[80], b[80];
cout << "Enter some input:\n";
cin >> a >> b;
cout << a << '-' << b << "END OF OUTPUT\n";
```

If the program runs as follows, what will be the coming output?

```
Enter some input:
The
     Midterm is easy.
```

# Exercise 3

Consider the following code
(and assume it is embedded in a complete and correct program and then run):

```
char myString[80];
cout << "Enter a line of input:\n";
cin.getline(myString, 6);
cout << myString << "<END OF OUTPUT";
```

If the program runs as follows, what will be the coming output?

```
Enter a line of input:
I really enjoy the course CS2311 in this semester!
```

# Today's Outline

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

# Common `cstring` Functions

| Function | Description |
|---|---|
| `strlen(str)` | returns the # of chars in a C string (before null-terminating character). |
| `strcmp(str1, str2)`,<br>`strncmp(str1, str2, n)` | compares two strings; returns 0 if identical, <0 if ***str1*** comes before ***str2*** in alphabet, >0 if ***str1*** comes after ***str2*** in alphabet. ***strncmp*** stops comparing after at most ***n*** characters. |
| `strchr(str, ch)`<br>`strrchr(str, ch)` | character search: returns a pointer to the first occurrence of ***ch*** in ***str***, or ***NULL*** if ***ch*** was not found in ***str***. `strrchr` find the last occurrence. |
| `strstr(haystack, n` | first occurrence of<br>ot found in ***haystack***. |
| `strcpy(dst, src)`,<br>`strncpy(dst, src, n)` | rminating character.<br>Assumes enough space in ***dst***. Strings must not overlap. **strncpy** stops after at most ***n*** chars, and does not add null-terminating char. |
| `strcat(dst, src)`,<br>`strncat(dst, src, n)` | concatenate ***src*** onto the end of ***dst***. **strncat** stops concatenating after at most ***n*** characters. Always adds a null-terminating character. |
| `strspn(str, accept)`,<br>`strcspn(str, reject)` | **strspn** returns the length of the initial part of ***str*** which contains only characters in ***accept***. **strcspn** returns the length of the initial part of ***str*** which does not contain any characters in ***reject***. |

Many string functions assume **valid cstring** input; i.e., ends in a null terminator.

# 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;
}
```

```
#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)
```

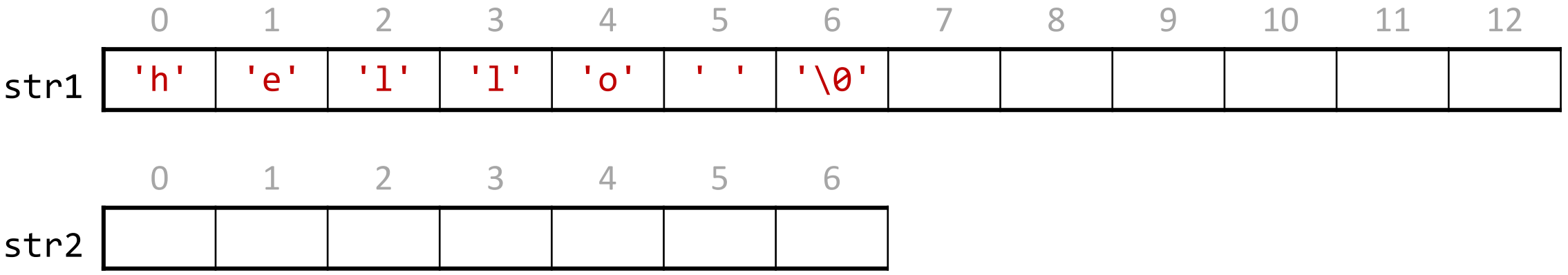
# strcat

- 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;
```

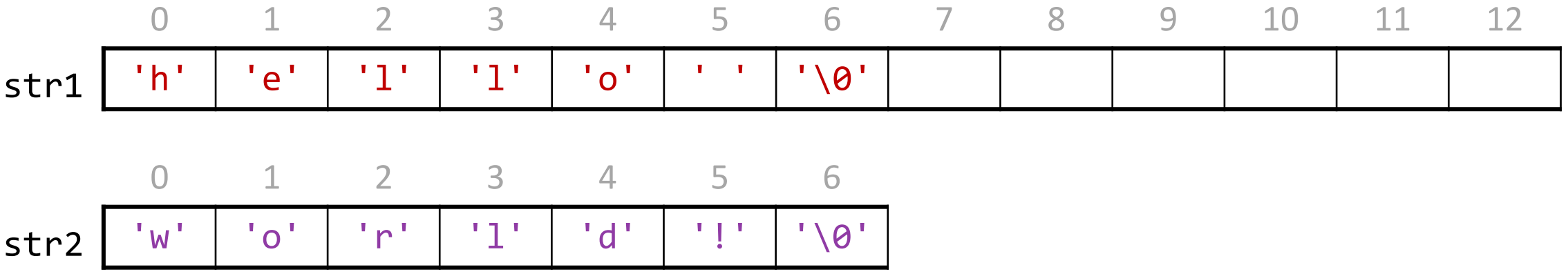
# strcat

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



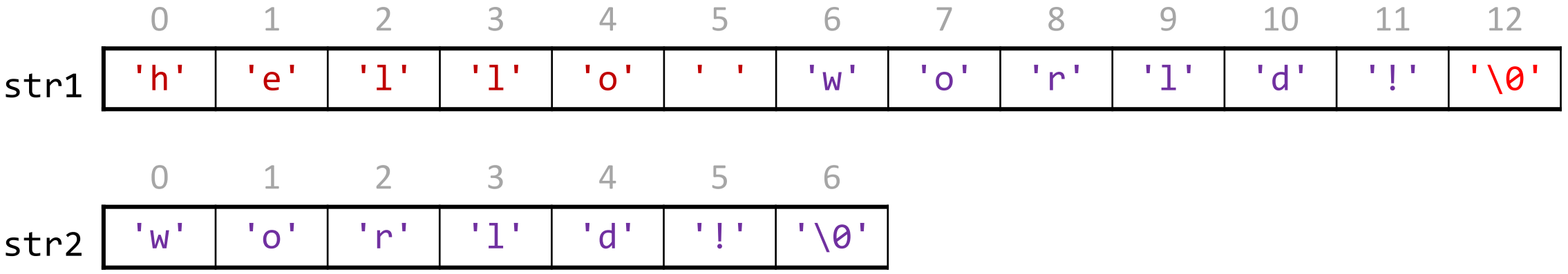
# strcat

```
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)

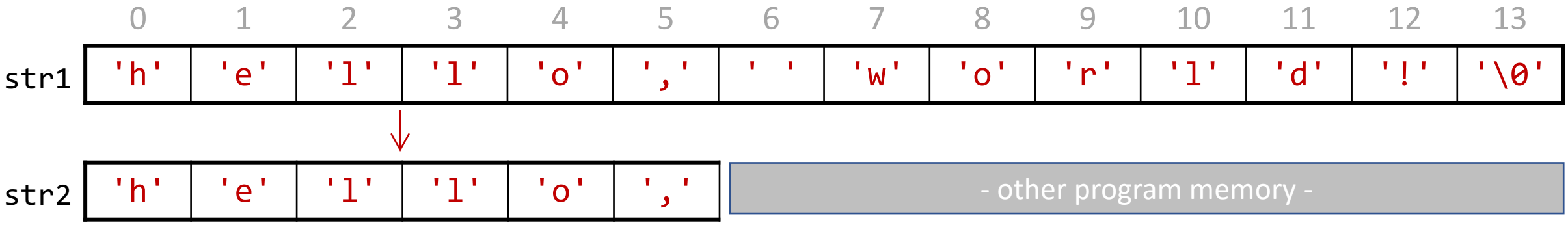
- **strspn(str, accept)**
  - returns the length of the initial part of ***str*** which contains only characters in ***accept***
- **strcspn(str, reject)**
  - returns the length of the initial part of ***str*** which does not contain any characters in ***reject***

```
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?

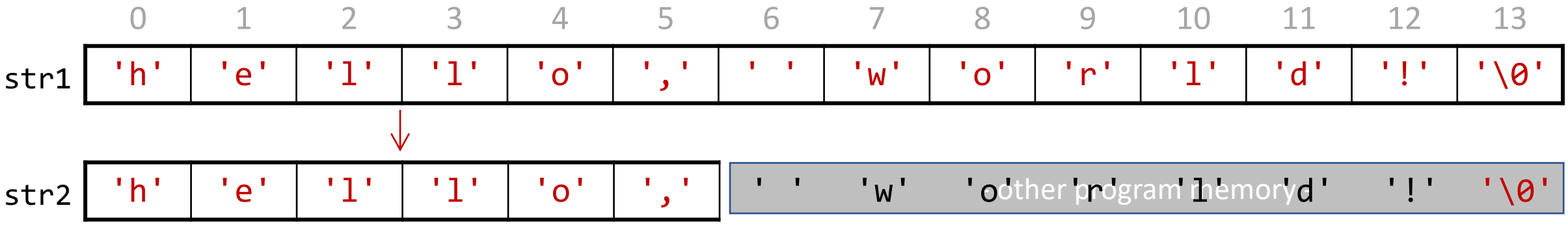
```
char str1[14];
strcpy(s, "hello, world!");
char str2[6];
strcpy(str2, str1);
```



# 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

# Exercise I

- What's printed out by the following program?

```
int main() {
   char str[9];
   strcpy(str, "Hi earth");
   str[2] = '\0';
   cout << "str=" << str << ", len=" << strlen(str);
   return 0;
}
```

# Exercise II

- Write a program to print a word backward
- Assume maximum input length is 20
- Example input/output
  - `hello`
  - `olleh`

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
   char                    ; // define an array with size 20
   int n;                    // length of str
   int i;
   cin >>                  ;
   n =                     ; // compute string length
   for (i =    ; i     ; i--)
      cout <<              ;
   return 0;
}
```

# Exercise II

- Write a program to print a word backward
- Assume maximum input length is 20
- Example input/output
  - `hello`
  - `olleh`

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
   char word[20];          // define an array with size 20
   int n;                  // length of str
   int i;
   cin >> word;
   n = strlen(word);       // compute string length
   for (i = n-1; i >= 0; i--)
      cout << word[i];
   return 0;
}
```

# Exercise III

- Write a program to let the user to input a line of string

- Reverse the case of the input characters and print the result
  - Lowercase characters are changed to uppercase
  - Uppercase characters are changed to lowercase

- Example input/output
  - `Hello World`
  - `hELLO wORLD`

# Exercise III

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
   char s[20];
   cin.getline(s, 20);
   for (int i = 0; s[i] != '\0'; i++) {
      if (                                    ) // uppercase letter
         cout <<                              ; // convert to lowercase
      else if (                               ) // lowercase letter
         cout <<                              ; // convert to uppercase
      else                                      // other letters
         cout <<                              ;
   return 0;
}
```

# Exercise III

```
#include <iostream>
#include <cstring>
using namespace std;
int main() {
   char s[20];
   cin.getline(s, 20);
   for (int i = 0; s[i] != '\0'; i++) {
      if (s[i] >= 'A' && s[i] <= 'Z')      // uppercase letter
         cout << char('a' + s[i]-'A');     // convert to lowercase
      else if (s[i] >= 'a' && s[i] <= 'z') // lowercase letter
         cout << char('A' + s[i]-'a');     // convert to uppercase
      else                                 // other letters
         cout << s[i];
   return 0;
}
```

# 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

# Exercise 1

- Which of the following are valid variable/constant names?

[A] you

[B] CityU_CS

[C] 2U

[D] $cake

[E] \you

[F] CityU-CS

# Exercise 2

```
int a = 0, b = 0;
cout << “b = “ << b << endl;

a = 0;
b = 1+(a++);
cout << “b= “ << b << endl;
cout << “a= “ << a << endl;

a = 0;
b = 1+(++a);
cout << “b= “ << b << endl;
cout << “a= “ << a << endl;
```

# Exercise 3

- what value will be printed?

```
int a = 0, b = 0;
bool x = (a!=0 && b=1);
cout << b << endl;
cout << x << endl;


bool y = (a!=0 || b=1);
cout << b << endl;
cout << y << endl;
```

# Exercise 4

What are the outputs?

```
int x = 10;
do
{
	cout << x++ << endl;
	x = x - 3;
} while (x > 0);
```

# Exercise 5

- What's the output of the following program?

```
void f(int y, int &x) {
   cout << "x=" << x++ << endl;
   cout << "y=" << y << endl;
}

void main(){
   int x=3, y=4;
   f(x, y);
   cout << x << " " << y << endl;
}
```

# Exercise 6

- Write a program: Given an integer *n*, return the number of ways you can write *n* as the sum of consecutive positive integers. Note that 1 <= n <= $10^9$

Example 1:

```
5
2
```

*Explanation: 5 = 2 + 3*

Example 2:

```
9
3
```

*Explanation: 9 = 4 + 5 = 2 + 3 + 4*

Example 3:

```
15
4
```

*Explanation: 15 = 8 + 7 = 4 + 5 + 6 = 1 + 2 + 3 + 4 + 5*