

Module 7 Guidance Notes

C++ Strings, C Characters & Strings, Recursion

ENGG1340

Computer Programming II

Before We Start

- We will deal with both C and C++ in this module, so please note the specific compiler settings when C is discussed.
- **C++:** We will be using the C++ 11 standard, so make sure that your compiler option is set appropriately. We suggest to use the following command to compile your C++ program:

```
g++ -pedantic-errors -std=c++11 your_program.cpp
```

- **C:** We will be using the C11 standard, so make sure that your compiler option is set appropriately. We suggest to use the following command to compile your C++ program:

```
gcc -pedantic-errors -std=c11 your_program.c
```

The C compiler will be used in Part II when we talk about C-Strings. We will have more details on the compilation environment then.

How to Use this Guidance Notes

- This guidance notes aim to lead you through the learning of the C/C++ materials. It also defines the scope of this course, i.e., what we expect you should know for the purpose of this course. (and which should not limit what you should know about C/C++ programming.)
- Pages marked with “Reference Only” means that they are not in the scope of assessment for this course.
- The corresponding textbook chapters that we expect you to read will also be given. The textbook may contain more details and information than we have here in this notes, and these extra textbook materials are considered references only.

How to Use this Guidance Notes

- We suggest you to copy the code segments in this notes to the coding environment and try run the program yourself.
- Also, try make change to the code, then observe the output and deduce the behavior of the code. This way of playing around with the code can help give you a better understanding of the programming language.

References

- cplusplus.com tutorial
 - [Strings](#)
- Textbook Chapters
 - [C++: How to program \(9th edition\)](#)
[Electronic version available from HKU library](#)
 - Ch. 6.20-22 (on recursion)
 - Ch. 21.1-9 (on C++ strings)

Topics

Part I: Strings (C++)

Part II: Characters and Strings in C

Part III: Recursion (C++)

Part I

STRINGS

Note that we will be using C++ in Part I

What are we going to learn?

- The `string` class as string representations
- String concatenation
- String comparison
- String I/O
- Member functions of the `string` class for string manipulation, e.g.,
 - `string::length()`
 - `string::empty()`
 - `string::substr()`
 - `string::find()`
 - `string::rfind()`
 - ...

The Class **String**

- C++ standard library provides a **class** (i.e., programmer defined data type) named **string** for more convenient handling of strings.
- We need to include the header file `string` to use the class `string`:

```
#include <string>
```

- A string object can be declared using the class name `string` and **initialized** with a C-string or another string

```
char a[80] = "Hello";           // a C-string
string msg1 = a;                // initialized with a C-string
string msg2 = "World";         // initialized with a string
literal
    string msg3 = msg1;         // initialized with a string
object
```

String Assignment

- The string class has its own end-of-string representation, for which we do not need to handle.
- Unlike C-string, we can initialize or change a string object using an assignment statement after its declaration:

Recall that in C-string, we need the null character '\0' to indicate end of string

```
char a[80] = "Hello";
string msg1, msg2, msg3;

msg1 = a;           // initialized with a C-string
msg2 = "World";     // initialized with a string
literal
msg3 = msg1;        // initialized with a string
object
```

String: Subscript Operator

- We may also access individual character using the subscript operator []:

```
string msg = "Hello  
World!";  
msg[11] = '?';  
  
cout << msg << endl;
```

Hello World?

Screen output

String Concatenation

- Two strings can be **concatenated** to form a longer string using the binary operator **+**

```
string msg1 = "I love ";  
string msg2 = "cats";  
string msg3 = msg1 + msg2;  
string msg4 = msg1 + "dogs";  
string msg5 = "I hate " + msg2 + " and  
dogs";
```

I love cats

I love dogs

I hate cats and dogs

- Note that at least one of the operands of **+** must be a string object.

```
string msg = "I love " + "dinosaur";
```

Here, both operands are string literals (i.e., constants)

String Comparison

- Strings can be compared lexicographically (dictionary order) using relational ($>$, $<$, $>=$, $<=$) and equality ($==$, $!=$) operators. The comparison is carried out in **a character by character manner**.

```
string msg1 = "Apple", msg2 = "apple";  
string msg3 = "apples", msg4 = "orange"
```

```
bool c1 = msg1 == msg2;  
bool c2 = msg1 < msg2;  
bool c3 = msg2 < msg3;  
bool c4 = msg3 != msg4;  
bool c5 = msg4 > msg3;
```

Note: at least one of the operands need to be a string object

c1

false

c2

true

c3

true

c4

true

c5

true

Example Programs

- A **palindrome** is a sequence of characters which reads the same forward or backward, e.g., “radar”, “level”, “kayak”, “madam”, “Was it a car or a cat I saw”.

Write a program to check if a string is a palindrome. You may hard-code the string in the program for the time being.

Hint: compare corresponding characters in the string

`palindrome.cpp`

- Write a program to reverse a string. E.g., the reverse of "apple" is "elppa". You may hard-code the string in the program for the time being.

Hint: use concatenation to construct the resulting string in reverse

`reverse_string.cpp`

I/O with String Objects

- Both `cout` and `cin` support string objects.
- The insertion operator `<<` and extraction operator `>>` work the same for string objects as for other basic data types

```
string msg;  
cin >> msg;  
cout << msg;
```

- Note that
 - The extraction operator `>>` **ignores whitespace** and stops reading when it encounters more whitespace
 - The word received by a string object will therefore **have any leading and trailing whitespace deleted**
 - Cannot read in a line or string that contains one or more blanks

I/O with String Objects

Example

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

int main()
{
    string word1, word2, word3;
    cout << "Please input a sentence: " << endl;

    cin >> word1 >> word2 >> word3;

    cout << "Word 1 = \" << word1 << "\"\n"
        << "Word 2 = \" << word2 << "\"\n"
        << "Word 3 = \" << word3 << "\"\n";
    return 0;
}
```

Please input a sentence:

I love dogs

Word 1 = "I"

Word 2 = "love"

Word 3 = "dogs"

Screen output

Use \" for a "
character in a string

string_io.cpp

How do we read in an entire line including spaces from the input then?

Reading a Line from Input

- We use the library function `getline()` to read in a line from the standard input and store the line in a string:

```
string s;  
cout << "Please input a sentence: " << endl;  
getline(cin, s);  
cout << "s = \" " << s << "\"\n";
```

string_getline.cpp

```
Please input a sentence:  
I    love        dogs  
s = "I    love        dogs"
```

Screen outputs

Reading a Line from Input

- The function `getline()` can be used to read in a line from the current position until a **delimitation character** is encountered

```
string s;  
cout << "Input 2 comma-separated phrases: " <<  
endl;  
getline(cin, s, ',');  
cout << "1st phrase = \"" << s << "\"\n";
```

string_getline.cpp

```
Input 2 comma-separated phrases:  
Stay hungry, stay foolish  
1st phrase = "Stay hungry"
```

Screen outputs

As you can see, without providing the third argument (',' in this case), the default delimitation character for the `getline` function is the newline character '\n'.

Member Functions

- The class string has a number of **member functions** which facilitate string manipulation, which includes
 - `string::length()` – returns length of the string
 - `string::empty()` – returns whether the string is empty
 - `string::substr()` – returns a substring
 - `string::find()` – finds the first occurrence of content in the string
 - `string::rfind()` – finds the last occurrence of content in the string
 - `string::insert()` – inserts content into the string
 - `string::erase()` – erases characters from the string
 - `string::replace()` – replaces part of the string

More member functions can be found at <http://www.cplusplus.com/reference/string/string>, but you are expected to be get familiar with the above functions only for this course.

string::length()

- Returns the **number of characters** in a string object

```
string s = "Stay hungry, stay foolish";  
int n = s.length();  
cout << "s has " << n << " characters. " <<  
endl;
```

Note we use `.` to invoke the member function of the string object `s`.

```
s has 25 characters.
```

Screen outputs

Exercise: Modify the reverse string and palindrome programs on [14. Example Programs](#) to work on any input words.

string::empty()

- Returns true if a string object is empty; false otherwise

```
string s;  
if (s.empty())  
    cout << "s is empty." << endl;  
else  
    cout << "s has " << s.length() << " characters."  
    \n";
```

What if s = " "?
Is this an empty string?

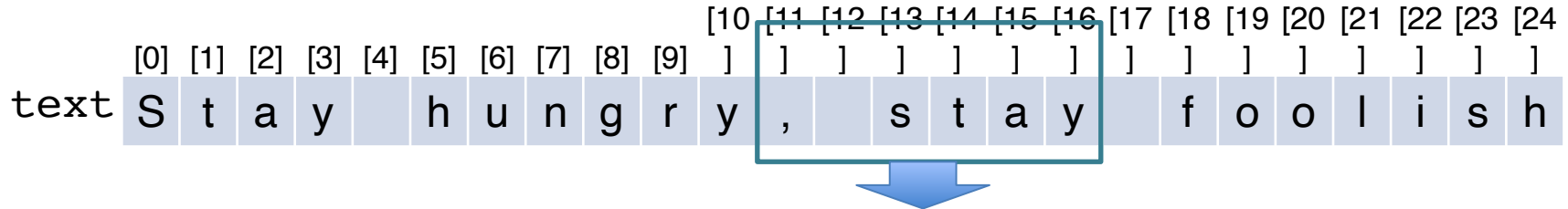
No, this is a string with a space character,
and its length is 1.

s is empty.

Screen outputs

string::erase()

- Erase **n** characters starting at a specific position **pos** from the current string
- Note that the string will be modified



`text.erase(11, 6)`

pos n

Resulting string:
"Stay hungry foolish"

Example

```
string firstName = "Alan";
string name = firstName + " Turing";
string str1 = "It is sunny. ";
string str2 = "";
string str3 = "C++ programming.";
string str4 = firstName + " is taking " + str3;

cout << str1.empty() << endl;
cout << str2.empty() << endl;
str3.erase(11,4);
cout << str3 << endl;
cout << firstName.length() << endl;
cout << name.length() << endl;
cout << str4 << endl;
```

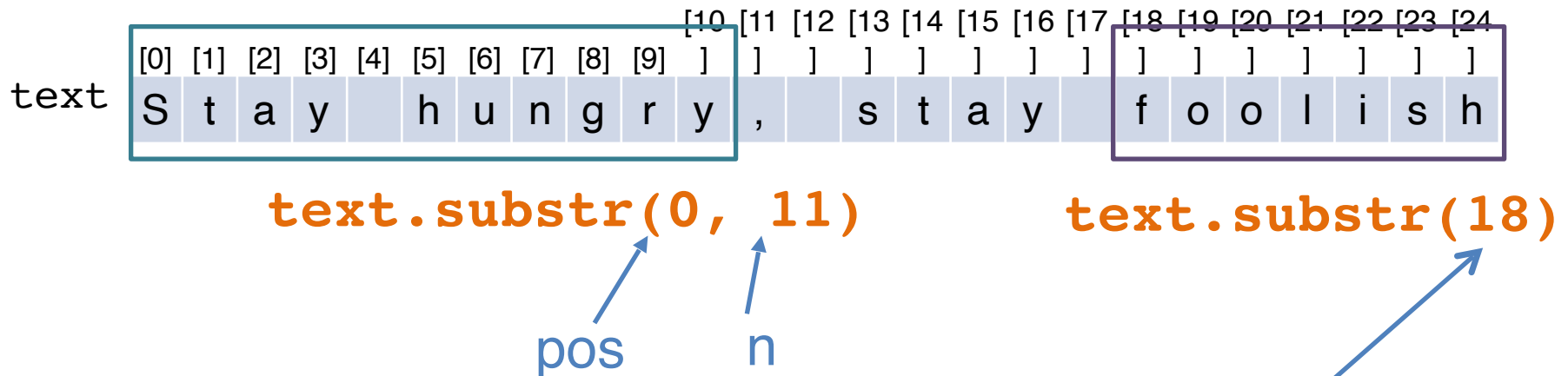
string_op.cpp

```
0
1
C++ program.
4
11
Alan is taking C++
programming.
```

Screen outputs

string::substr()

- Returns a **substring** of the current string object starting at the character position **pos** and having a length of **n** characters



The second parameter is omitted, this extracts a substring till the end of string.

string::substr()

Example

```
string s;  
string str;  
  
s = "It is cloudy and warm.";   
  
cout << s.substr(0, 5) << endl;  
cout << s.substr(6, 6) << endl;  
cout << s.substr(6, 16) << endl;  
cout << s.substr(17, 10) <<  
endl;  
cout << s.substr(3, 6) << endl;  
str = s.substr(0, 8);  
cout << str << endl;  
str = s.substr(2, 10);  
cout << str << endl;
```

Screen outputs

```
It is  
cloudy  
cloudy and warm.  
warm.  
is clo  
It is cl  
is cloudy
```

substring.cpp

string::find()

- Searches a string object for a given string `str`, and returns the position of the first occurrence

```
find(str)
```

- When `pos` is specified the search only includes characters at or after position `pos`, ignoring any possible occurrences in previous locations

```
find(str, pos)
```

- If there is no occurrence of `str`, the constant value `string::npos` (i.e., `-1`) will be returned

string::find()

Example

```
string s = "Outside it is cloudy and  
warm.";  
string t = "cloudy";  
  
cout << s.find("is") << endl;  
cout << s.find('s') << endl;  
cout << s.find(t) << endl;  
cout << s.find('i', 6) << endl;  
cout << s.find('o') << endl;  
if (s.find("the") == -1)  
    cout << "not found" << endl;  
if (s.find("the") == string::npos)  
    cout << "not found" << endl;
```

string_find.cpp

This example shows that
the search is case-sensitive

Screen outputs

```
11  
3  
14  
8  
16  
not found  
not found
```

string::rfind()

- Searches the current string object for the content specified in `str`, and returns the position of **the last occurrence**

```
rfind(str)
```

This is essentially to search in the reverse direction from the end of the string

- When `pos` is specified the search only includes characters at or before position `pos`, ignoring any possible occurrences in later locations

```
rfind(str, pos)
```

- If there is no occurrence of `str`, the constant value **`string::npos`** (i.e., `-1`) will be returned

string::rfind()

Example

```
string s = "Outside it is cloudy and  
warm.";
string t = "cloudy";

cout << s.rfind("is") << endl;
cout << s.rfind('s') << endl;
cout << s.rfind(t) << endl;
cout << s.rfind('i', 6) << endl;
cout << s.rfind('o') << endl;
if (s.rfind("the") == -1)
    cout << "not found" << endl;
if (s.rfind("the") == string::npos)
    cout << "not found" << endl;
```

string_rfind.cpp

Screen outputs

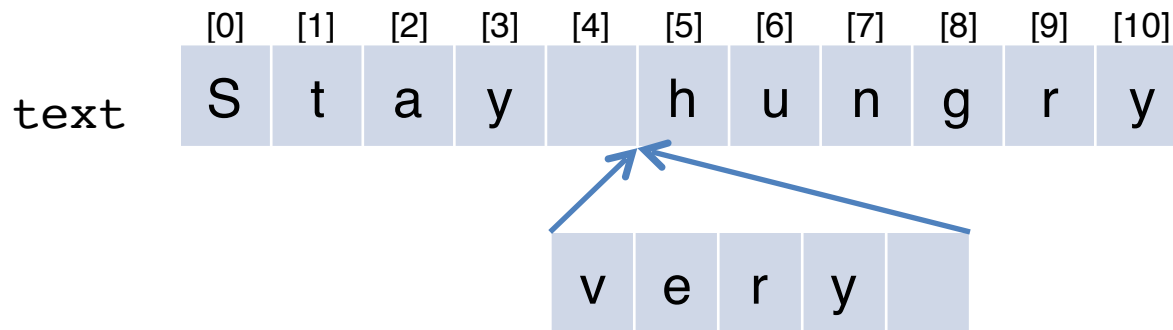
```
11
12
14
4
16
not found
not found
```

Programming Exercises

- Write a program that finds the positions of ALL occurrences of a substring in a string, starting from the first occurrence to the last occurrence.
- Write a program that finds the positions of ALL occurrences of a substring in a string, starting from the last occurrence to the first occurrence.

string::insert()

- Inserts the content specified in `str` at position `pos` of the current string



`text.insert(5, "very ")`

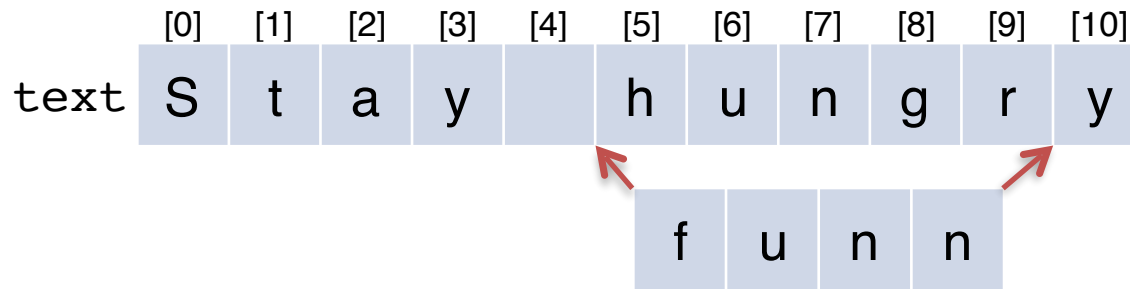
`pos`

`str`

Resulting string: "Stay very hungry"

string::replace()

- Replaces `n` characters starting at position `pos` from the current string by the content specified in `str`



`text.replace(5, 5, "funn ")`

pos n str

Resulting string: "Stay funny"

Example

```
string s1 = "Cloudy and warm.";
string s2 = "Angel is taking
programming.";
string t1 = " very";
string t2 = "Nelson";

cout << s1.insert(10, t1) << endl;
cout << s2.replace(0, 5, t2) << endl;
```

string_insert_replace.cpp

Cloudy and very warm.
Nelson is taking
programming.

Screen outputs

Tutorial Problems - Strings

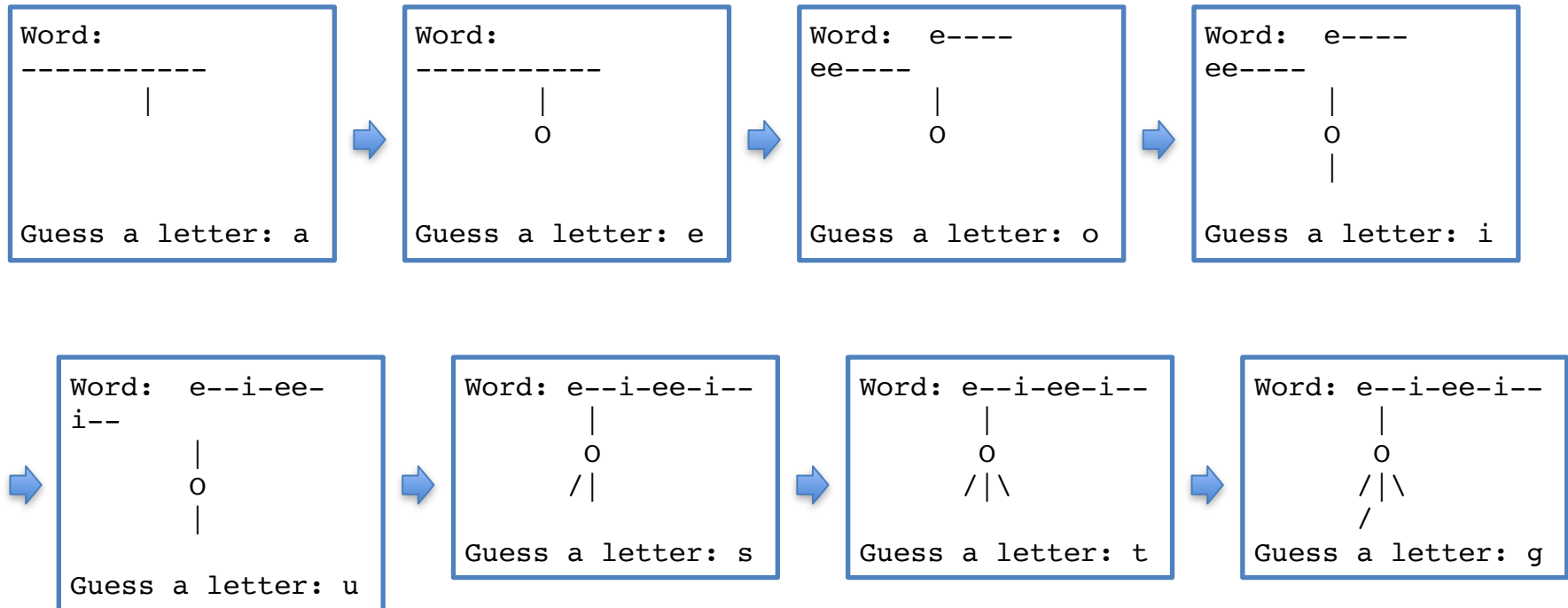
THE HANGMAN GAME

The Hangman Game

- You are going to implement the Hangman game in this task.
- Your program will generate a random word for a user to guess.
- Your program should display dashes for unrevealed letters for the word
- The player will guess a letter in each round
 - If the letter appears in the word, display all occurrences of the letter in the word
 - Otherwise add one stroke to the hangman picture
- The game ends when
 - Either the player wins by successfully guessing the complete word
 - Or the player loses when the hangman picture is shown in full

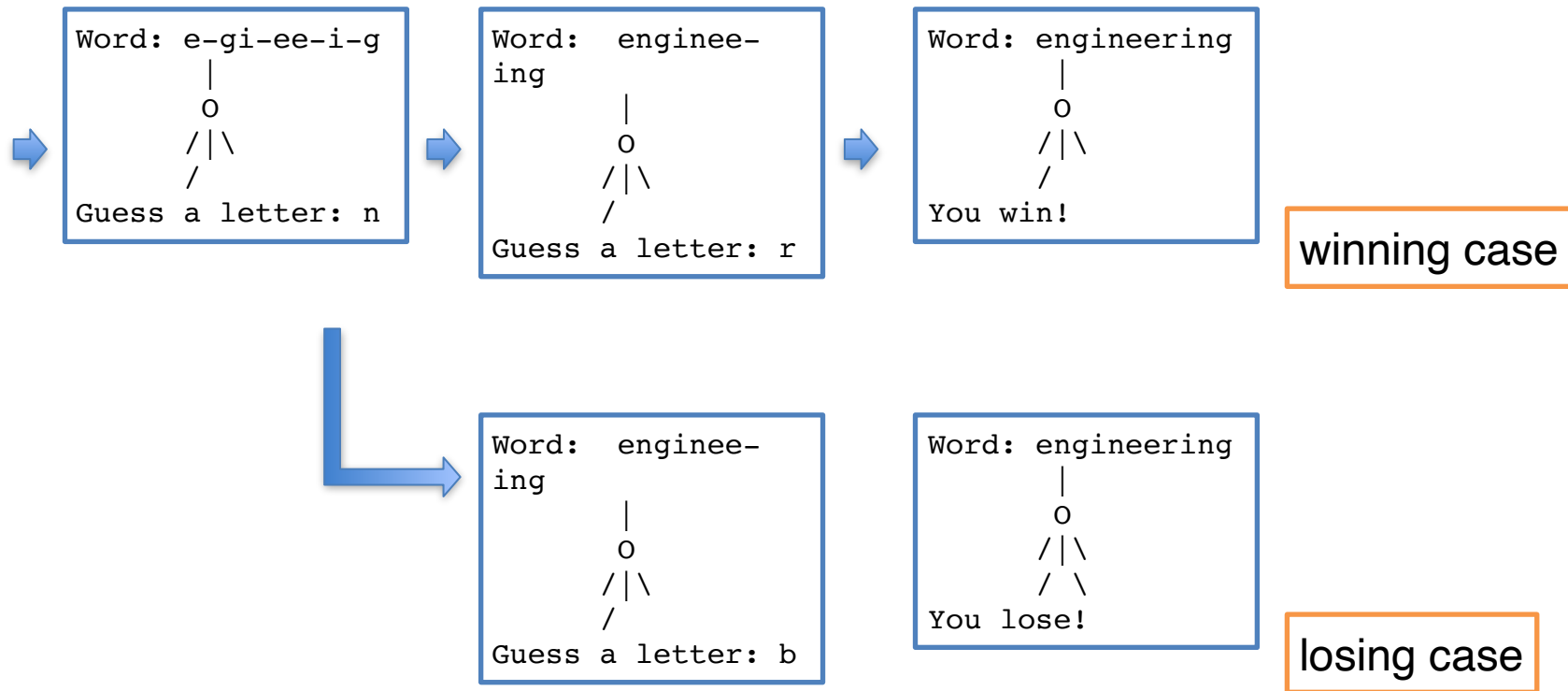
The Hangman Game

- Sample game play (suppose the word is “engineering”):



The Hangman Game

- Sample game play (suppose the word is “engineering”):



hangman.cpp provides the completed version of this tutorial problem.
You may compile and run it to see the expected results first.

Task 1: Generate a random word

- Open `hangman_incomplete.cpp`.
- The word list to be guessed by the player is stored in an array of strings:

```
string wordlist[10] = { "engineering", "hangman", "brainstorm",  
                        "random",  
                        "envelope", "interface", "iceberg", "humour", "lemon",  
                        "commander"};
```

- Generate a random number to randomly select one of the words in the word list as the guessing word.

```
// TODO: Task 1: fill in the index to choose a random word from the  
word list  
string word = wordlist[ ];
```

Store the guessing word

Complete this in the
main() function

Task 2: Initialize the answer

- Next, we need to initialize the answer string so that it contains only the hyphen '-' character to hide all letters of the guessing word. E.g., if the guessing word is "random", the answer is initialized to "-----".
- Task 2 is to complete the function `init_answer()`.

```
// return a string containing a sequence of '-' with specified length
```

```
string init_answer( int length )
```

```
{
```

```
    string g;
```

```
    // TODO: Task 2: compose the string g so that its length = length and
```

```
    // it only contains the letter '-'
```

```
    return g;
```

```
}
```

Complete the task here.

Idea: Append the letter '-' to g
until g is of the required length

Task 3: Call init_answer()

- In the main function, we will need to call `init_answer()` and store the resulting string in the variable `answer`.

```
// initialize the answer string  
// TODO: Task 3: call init_answer and assign it to answer  
string answer =
```

Complete this by calling the function
`init_answer()`.

What is the parameter that needs to be
passed?

What should be the length of the string
`answer`?

Game Logic

Now take a look at the while loop in the main() function.

The game logic is as follows:

- While the game is not ended
 1. Show the current answer and the hangman figure
 2. Determine whether it is end of game. There are two ways to end a game:
 - a. the player guessed the word. In this case, all the letters in the string answer should be revealed.
 - b. the player loses because he made too many wrong guesses.
 3. If it is the end of game, print the appropriate message and quit
 4. Otherwise, ask the user to make a guess.
 - a. If the guess is correct, update the answer by unmasking the correct letters
 - b. Otherwise, update the counter for recording the number of wrong guesses

Task 4: Game Logic

- Let's complete the game logic in the main function().

```
while (!endgame)
{
    cout << "Word: " << answer << endl;
    show_hangman( num_wrong_guess );

    // determine if it's an end game
    // TODO: Task 4: fill in the end-of-game
    conditions
    if ( )
    {
        cout << "You win!" << endl;
        endgame = true;
    }
    else if ( )
    {
        cout << "You lose!" << endl;
        endgame = true;
    }
    else {
        ...
    }
}
```

Fill in the end-of-game conditions
here

After guessing a letter

- After the player has guessed a letter, we need to check if the letter is in the guessing word.

```
while (!endgame)
{
    ...
    else {

        cout << "Guess a letter: ";
        cin >> guess;

        if ( isGuessInWord( word, guess) )
            update_answer( word, answer,
guess );
        else
            num_wrong_guess++;
    }
}
```


This part in the main() function is done for you already.

Check whether the letter input by the player is in the guessing word

If yes, update the current answer to unmask the letter
e.g., if word is "apple", the answer before the guess is "-----" and the player inputs the letter 'p', then answer would become "-pp--"

Task 5: Check if a letter is in a word

- Complete the function `isGuessInWord()` which checks if a letter appears in a word

```
// return if the letter c appears in the string w
bool isGuessInWord( string w, char c)
{
    // TODO: Task 5: complete the function return a boolean value
    // to indicate whether c appears in w
    
}
```

Complete the function here.
CHALLENGE: Can you do this by using only one statement?

Task 6: Unmask the correct letter

- If the player guesses a letter correctly, we will need to unmask it in the string answer. This is done in the function `update_answer()`.

```
// assume that w and ans are of the same length
// copy all occurrences of letter c in string w to the
// corresponding
// positions in string ans
// e.g. string w = "xyzxyz"
// and string ans = "-----", char c = 'y'
// then ans will become "-y--y-"
void update_answer( string w, string & ans, char c)
{
    // TODO: Task 6
}
```

Note that `ans` is **passed by reference**, which means that any modification to `ans` will be reflected to the corresponding actual parameter passed to this function

Complete the function here.
Idea: Go through the letters in `w` one by one, and whenever the letter `c` is found, update the corresponding position in `ans`

Task 7: Display the hangman figure



- Finally, complete the `show_hangman()` function which draws the hangman figure depending on how many times the player has made a wrong guess.

You should be able to play the hangman game now!

```
// show the hangman diagram
// state is the number of wrong guess
void show_hangman( int state )
{
    // first line
    cout << "    |" << endl;

    // second line
    if (state >= 1)
        cout << "    O" << endl;
    else
        cout << endl;

    // TODO: Task 7: Complete displaying the 3rd and 4th lines
    // of the hangman figure

    // third line
    
    // fourth line
    

}
```

Complete these two parts to draw the 3rd and 4th lines of the hangman figure.
Note that you need to write `\\` to output the character `\`

Now get ready to
write your first C program

Part II

CHARACTERS AND STRINGS IN C

What are we going to learn?

- C program compilation
- A simple program in C
- C basic standard I/O
 - printf
 - scanf
- Character functions
 - isdigit
 - tolower
 - atoi
 - ...
- String functions
 - strcpy
 - strcmp
 - ...

Compiling a C Program

Although we can use C-strings in a C++ program (like what we did in Module 6, **we will start writing C programs** and play with C-strings and its I/O in C. In order to do this, we will be using the C compiler (C11 version).

We may still compile and execute a C program using **the Atom editor** with gcc-make-run program, just **make sure that**:

- Your source program file has a name with an extension .c, e.g., program1.c
- In the Atom editor, in “Preferences” -> “Packages” -> “Gcc Make Run” -> “Settings” and under “Compiler Flags”, put down “**-pedantic-errors -std=c11**” instead (see next page).

For **command line compilation**, use:

gcc -pedantic-errors -std=c11 your_program.c -o your_program

You can then use “./your_program” to execute the program.

⚙ Settings

gcc Compiler

Compiler for `C`, in full path or command name (make sure it is in your `$PATH`)

Default: `gcc`

g++ Compiler

Compiler for `C++`, in full path or command name (make sure it is in your `$PATH`)

Default: `g++`

make Utility

The `make` utility used for compilation, in full path or command name (make sure it is in your `$PATH`)

Default: `make`



Unconditional Build

Will not check if executable is up to date

Compiler Flags

Flags for compiler, eg: `-Wall`

`-pedantic-errors -std=c11`

Hello World in C

Take a look at this simple program in C.

```
/* this is my first C hello world program */
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

helloworld.c

Compare this to the C++ program

```
// this is my first C++ hello world program
#include <iostream>
using namespace std;

int main() {
    cout << "Hello World!" << endl;
    return 0;
}
```

Hello World in C

```
/* this is my first C hello world program */  
#include <stdio.h>  
  
int main() {  
    printf("Hello World!\n");  
    return 0;  
}
```

Comments are enclosed in
/* ... */
Comments may span multiple
lines

Include **stdio.h** for using
standard I/O functions such as
printf

Standard output are done via
the **printf** statement

The newline character '**\n**' is
used for the line break.

Try compile the program and check the result.

Adding Two Numbers

addition.c

```
#include <stdio.h>

int main()
{
    int x, y, z;

    printf("Enter first integer:\n");
    scanf("%d", &x);

    printf("Enter second integer:\n");
    scanf("%d", &y);

    z = x + y;
    printf("Sum is %d\n", z);

    return 0;
}
```

Standard input is done via the scanf function. "%d" is the format control string which indicates that the input data should be in integer.

&x is the memory location of the variable (x) in which the data should be stored.

This statement essentially reads in an integer and store it to the variable x.

Screen output

```
Enter first integer:
32
Enter second integer:
65
Sum is 97
```

The numbers in blue above are user inputs.

We will talk more about formatted input and output in C in later modules.

Character Handling Functions

Recall (from Module 6) that a character takes up 1 byte of storage space. The values 0-127 stored in a char correspond to a particular character in the ASCII character set. We may also perform arithmetic operations on char as if they are of int data type.

The <ctype.h> header file contains handy functions for character handling. Here are some examples:

<code>int isdigit(int c)</code>	Returns a nonzero (true) value if c is a digit, and 0 (false) otherwise
<code>int isalpha(int c)</code>	Returns a nonzero (true) value if c is a letter, and 0 (false) otherwise
<code>int isalnum(int c)</code>	Returns a nonzero (true) value if c is a digit or a letter, and 0 (false) otherwise
<code>int islower(int c)</code>	Returns a nonzero (true) value if c is a lowercase letter, and 0 (false) otherwise
<code>int isupper(int c)</code>	Returns a nonzero (true) value if c is an uppercase letter, and 0 (false) otherwise
<code>int toupper(int c)</code>	If c is a lowercase letter, returns c as an uppercase letter. Otherwise, returns the argument unchanged.
<code>int tolower(int c)</code>	If c is an uppercase letter, returns c as lowercase letter. Otherwise, returns the argument unchanged.

Reference only: check [this](#) for more character handling functions

charfunc.c

"%c%s%s" indicates that printf will output a character (%c) followed by two strings (%s) to the standard output.

```
#include <stdio.h>
#include <ctype.h>
```

```
int main()
{
    char a;
```

The first character to output is stored in the variable a

The next string to output depends on the value of isdigit(a), if it is true, then " is " is output; otherwise " is not " is output

The last string to output is "a digit"

```
    a = '7';
    printf("%c%s%s\n", a, isdigit(a) ? " is " : " is not ", "a digit" );
    a = '$';
    printf("%c%s%s\n", a, isdigit(a) ? " is " : " is not ", "a digit" );

    a = 'B';
    printf("%c%s%s\n", a, isalpha(a) ? " is " : " is not ", "a letter" );
    a = 'b';
    printf("%c%s%s\n", a, isalpha(a) ? " is " : " is not ", "a letter" );
    a = '4';
    printf("%c%s%s\n", a, isalpha(a) ? " is " : " is not ", "a letter" );

    a = 'Z';
    printf("%c%s%s\n", a, islower(a) ? " is " : " is not ", "a lowercase letter" );
    a = 'z';
    printf("%c%s%s\n", a, isalpha(a) ? " is " : " is not ", "a lowercase letter" );
    a = '5';
    printf("%c%s%s\n", a, isalpha(a) ? " is " : " is not ", "a lowercase letter" );

    a = 'M';
    printf("%c%s%s\n", a, islower(a) ? " is " : " is not ", "an uppercase letter" );
    a = 'm';
    printf("%c%s%s\n", a, isalpha(a) ? " is " : " is not ", "an uppercase letter" );
    a = '#';
    printf("%c%s%s\n", a, isalpha(a) ? " is " : " is not ", "an uppercase letter" );

    return 0;
}
```


Character Handling Functions

Screen output of charfunc.c

```
7 is a digit
$ is not a digit
B is a letter
b is a letter
4 is not a letter
Z is not a lowercase letter
z is a lowercase letter
5 is not a lowercase letter
M is not an uppercase letter
m is an uppercase letter
# is not an uppercase letter
```

Character Handling Functions

charconvert.c

```
#include <stdio.h>
#include <ctype.h>

int main()
{
    char a;

    a = 'e';
    printf("%c converted to uppercase is %c\n", a, toupper(a));
    a = 'S';
    printf("%c converted to uppercase is %c\n", a, toupper(a));
    a = '%';
    printf("%c converted to uppercase is %c\n", a, toupper(a));
    a = '9';
    printf("%c converted to uppercase is %c\n", a, toupper(a));

    a = 'w';
    printf("%c converted to lowercase is %c\n", a, tolower(a));
    a = 'R';
    printf("%c converted to lowercase is %c\n", a, tolower(a));
    a = '&';
    printf("%c converted to lowercase is %c\n", a, tolower(a));
    a = '2';
    printf("%c converted to lowercase is %c\n", a, tolower(a));

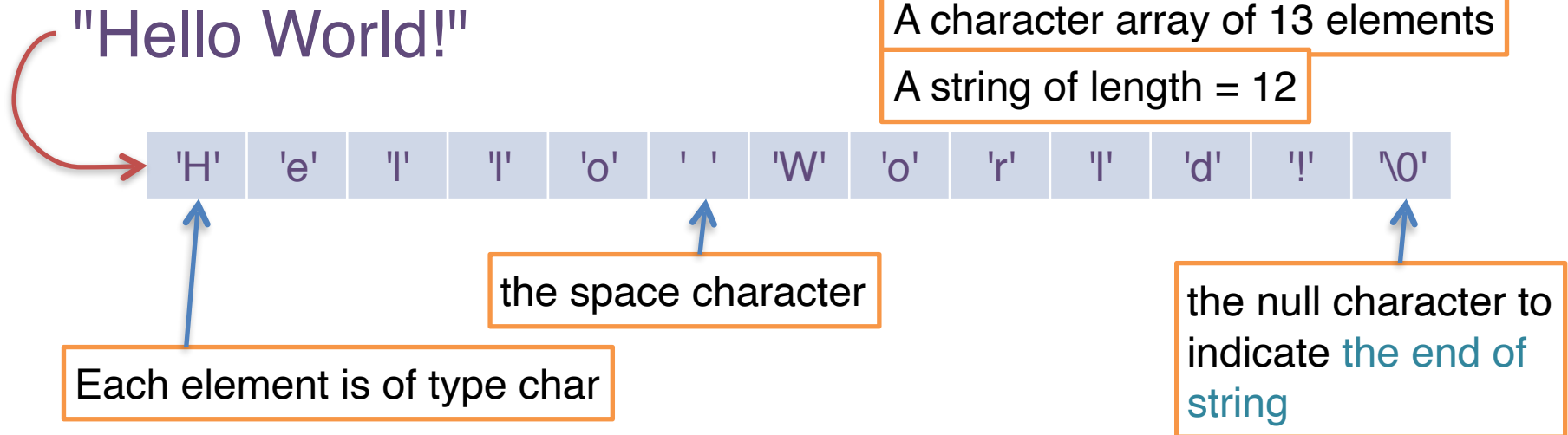
    return 0;
}
```

Screen output

```
e converted to uppercase is E
S converted to uppercase is S
% converted to uppercase is %
9 converted to uppercase is 9
w converted to lowercase is w
R converted to lowercase is r
& converted to lowercase is &
2 converted to lowercase is 2
```

Strings in C

Recall from Module 6 that a string can be represented as an array of char (C-Strings), which is ended by a null character (`'\0'`):



We do not have string objects in C, and **C-Strings are the only representation for strings in a C program.**

String Handling Functions

You may manipulate individual characters in a char array storing a string, or you may make use of some string handling functions.

Let's take a look at some **string conversion functions** first. Include **<stdlib.h>** for using the following functions

<code>int atoi(const char *str)</code>	Converts the string pointed to by str to int
<code>double atof(const char *str)</code>	Converts the string pointed to by str to double

the "const" specifier indicates that the parameter str (i.e., the string input to these functions) will not be modified by the functions

the "*" here means a pointer to the input string (i.e. an address of the memory location where the string is stored). For the time being, **just remember that the name of the character array storing the string provides such a pointer**. See example next page on the usage.

String Conversion Functions

stringconvert.c

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i;
    double d;
    long l;

    i = atoi("1340");
    printf("The string \"1340\" converted to int is %d\\n", i);
    printf("The converted value minus 111 is %d\\n", i-111);

    d = atof("23.9");
    printf("The string \"23.9\" converted to double is %.3f\\n", d);
    printf("The converted value divided by 3 is %.3f\\n", d / 3);

    return 0;
}
```

the "%.3f" specifier indicates that the parameter printf is to output a floating point number (f) with 3 decimal places (.3)

Screen output

```
The string "1340" converted to int is 1340
The converted value minus 111 is 1229
The string "23.9" converted to double is 23.900
The converted value divided by 3 is 7.967
```

String Handling Functions

Include `<string.h>` for using the following functions

<code>char *strcpy(char *s1, const char *s2)</code>	Copies the string s2 into the array s1
<code>char *strncpy(char *s1, const char *s2, size_t n)</code>	Copies n characters of the string s2 into the array s1
<code>char *strcat(char *s1, const char *s2)</code>	Appends the string s2 into array s1. The first character of s2 overwrites the terminating '\0' character of s1
<code>char *strncat(char *s1, const char *s2, size_t n)</code>	Appends n characters of the string s2 into array s1. The first character of s2 overwrites the terminating '\0' character of s1
<code>int strcmp(const char *s1, const char *s2)</code>	Compares the string s1 to the string s2. Returns 0, less than 0, or greater than 0 if s1 is equal to, less than, or greater than s2, in lexicographical order, respectively
<code>size_t strlen(const char *s)</code>	Returns the length of string s (i.e., the number of characters preceding '\0')

This is the same as the unsigned integer type, i.e., the number must be a non-negative number

Note that C-strings do not support direct assignment, e.g., you CANNOT write

```
char s1[20];  
s1 = "abc";
```

so you need `strcpy()` to do assignment

Also, comparison of strings using `==` is not supported, you'll need to use `strcmp()` for C-strings comparison

String Handling Functions

```
#include <stdio.h>
#include <string.h>

int main()
{
    char x[30] = "ENGG1340 computer programming";
    char y[30], z[10];
    char s1[20] = "Keep calm ";
    char s2[20] = "and code ";
    char s3[40] = "";

    strcpy(y, x);
    printf("%s\n%s\n", x, y);

    strncpy(z, x, 8);
    printf("%s\n%s\n", z, y);

    printf("%s\n", s1);
    strcat(s1, s2);
    printf("%s\n", s1);
    strncat(s3, s2, 4);
    printf("%s\n", s3);
    strcat(s3, s1);
    printf("%s\n\n", s3);

    printf("%d\n", strcmp(x, y));
    printf("%d\n", strcmp(s1, s3));
    printf("%d\n\n", strcmp(s1, x));

    printf("length of x = %d\n", strlen(x));

    return 0;
}
```

Note that you'll need to make sure that the destination char array for the copy functions is large enough for the resulting string; otherwise, runtime error will occur

Screen
output

```
ENGG1340 computer programming
ENGG1340 computer programming
ENGG1340
ENGG1340 computer programming
Keep calm
Keep calm and code
and
and Keep calm and code

0
-22
6
29
```

string_manipulation.c

Using C-Strings in C++

You may use C-strings in C++, and to use the character and string handling functions, include the following headers

in C	in C++
<code>#include <ctype.h></code>	<code>#include <cctype></code>
<code>#include <stdlib.h></code>	<code>#include <cstdlib></code>
<code>#include <string.h></code>	<code>#include <cstring></code>

To convert a C++ string to C-string, you may use the `.c_str()` member function of a string object:

```
#include <string>
#include <cstring>

int main()
{
    string str = "abc";
    char s1[10];

    strcpy(s1, str.c_str());

    cout << s1 << endl;

    return 0;
}
```


Now switching back to C++...

Remember to set your compiler options properly before you proceed.

Part III

RECURSION

What are we going to learn?

- Recursive definition
- Recursive functions in C++
- Flow of control in recursive functions
- General structure of a recursive function
- Examples of recursive functions
- Stack overflow problem
- Recursion versus iteration

Recursive Definition

- Some problems are **recursive** by nature, i.e., it has a **recursive definition** which means that the problem can be defined in terms of a smaller version of itself.

Consider the factorial of a nonnegative integer:

Definition 1

$$0! = 1$$

$$n! = n \times (n-1) \times (n-2) \times \dots \times 2 \times 1, \quad \text{if } n > 0$$

An iterative definition

Definition 2

$$0! = 1$$

$$n! = n \times (n-1)!, \quad \text{if } n > 0$$

A recursive definition

Recursive Definition

- How does a recursive definition work?

Base case

$$\text{Eq. (1): } 0! = 1$$

General case

$$\text{Eq. (2): } n! = n \times (n-1)!, \quad \text{if } n > 0$$

To calculate $3!$:

1. Apply Eq. (2) : $3! = 3 \times 2!$ General case

2. Apply Eq. (2) : $2! = 2 \times 1!$ General case

3. Apply Eq. (2) : $1! = 1 \times 0!$ General case

4. Apply Eq. (1) : $0! = 1$ Base case

7. Substitute: $3! = 3 \times 2 = 6$

6. Substitute: $2! = 2 \times 1 = 2$

5. Substitute: $1! = 1 \times 1 = 1$

Recursive Definition

- Properties for a recursive definition
 - Must have one (or more) base cases
 - The general case must be reduced to a base case eventually
 - The base case terminates the recursion
- Some more examples of recursive problems
 - Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
 - $F_n = F_{n-1} + F_{n-2}$, $F_0 = 0$, $F_1 =$
 - Tower of Hanoi

General case: a number is the sum of its
previous two numbers

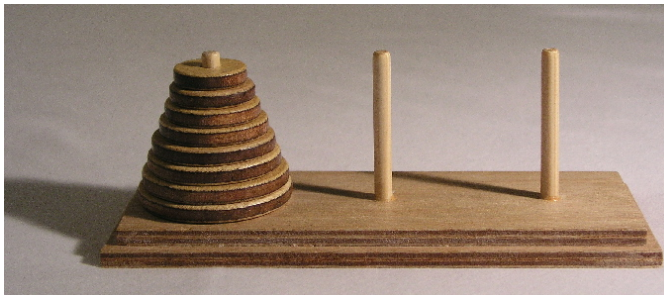


Image from Wikimedia Commons

Recursive Function

- In C/C++, we may write **recursive function** to implement recursion.
- A recursive function is one that **contains a call to itself**.

Base case

General case

```
int factorial(int num)
{
    if (num == 0)
        return 1;
    else
        return num * factorial(num - 1);
}
```

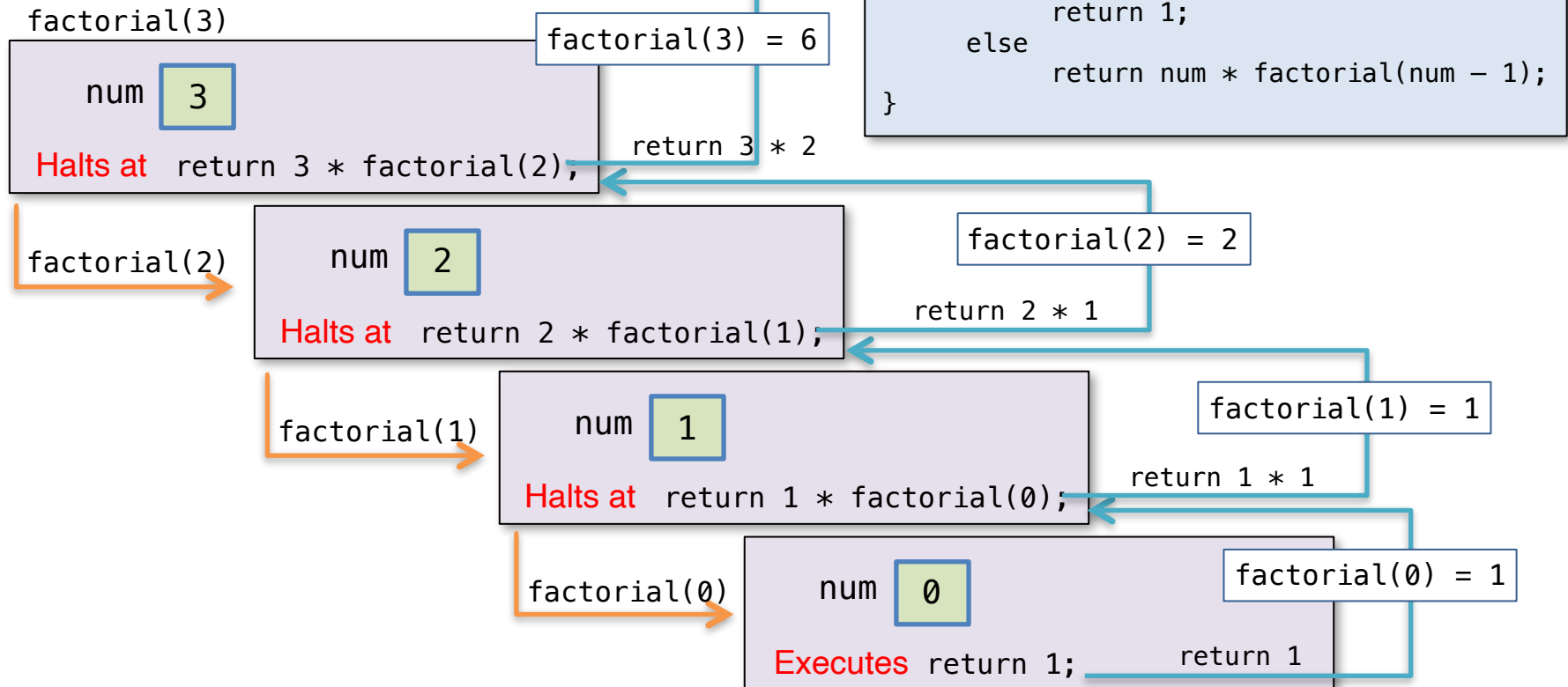
factorial.cpp

Since the argument passed to the functions keeps decrementing by 1, we are certain that the base case will be reached eventually which stops the recursion.

Flow of Control

- Flow of control is essentially the same as function calls, except that the same function is repeatedly called.

Suppose we call `factorial(3)`:



General Structure

- The process of calling a function itself recursively can be repeated any number of times.
- How to avoid **infinite recursion**?
- General structure for a recursive function definition:
 - Having one or more recursive calls to itself to accomplish smaller tasks
 - Having one or more base cases **without using recursive calls** to terminate the recursion

```
int factorial(int num)
{
    if (num == 0)
        return 1;
    else
        return num * factorial(num - 1);
}
```

Base case without using recursion

Recursion to handle smaller tasks by making recursive calls

Example: Fibonacci Sequence

Recursive definition for the problem:

$$F_0 = 0, F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, \text{ if } n > 1$$

Base case

Recursion

The sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34,

...

```
int fib(int num)
{
    if (num < 2)
        return num;
    else
        return fib(num-1) + fib(num-2);
}
```

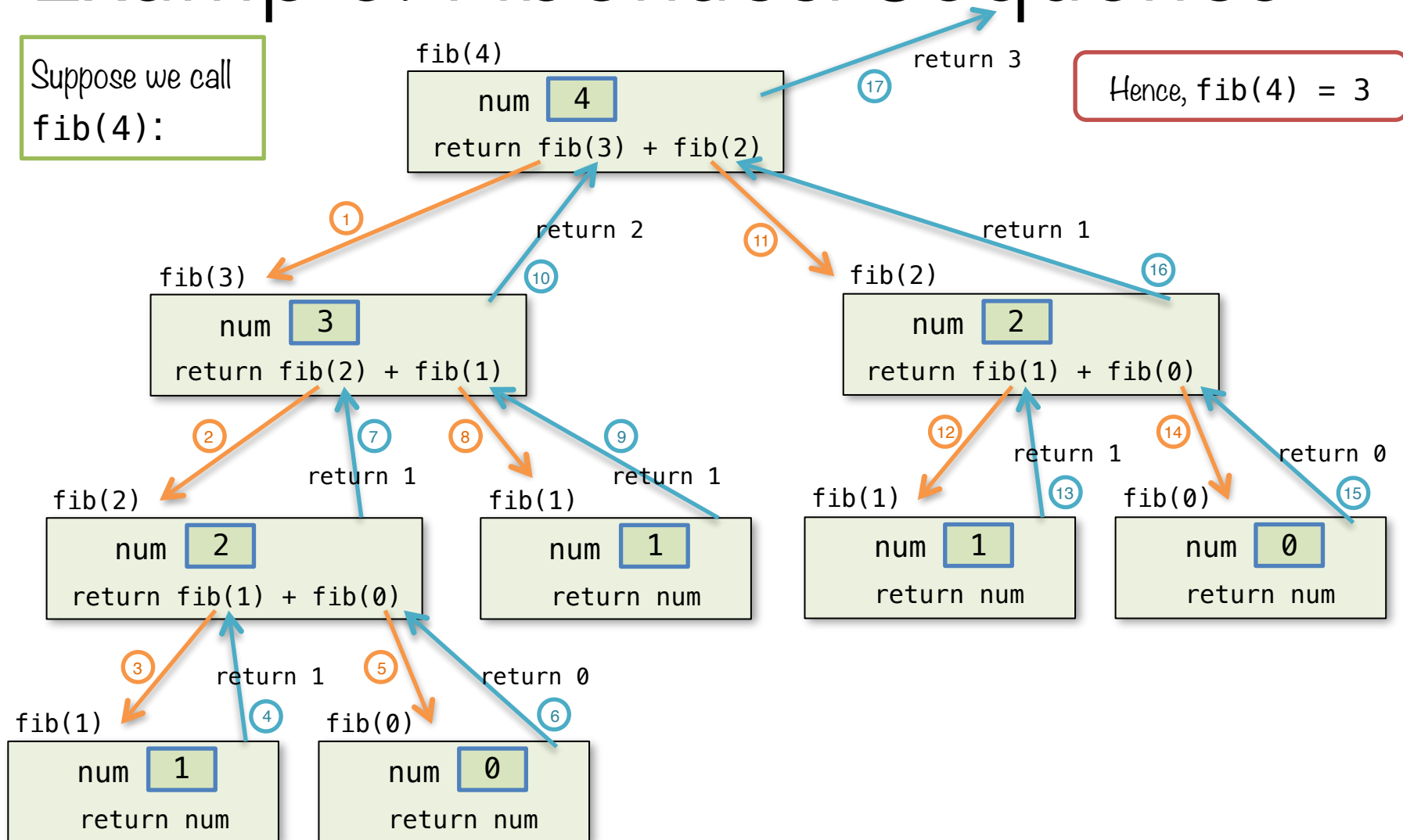
Base case without using recursion

Recursion to handle smaller tasks by making recursive calls

fibonacci.cpp

Example: Fibonacci Sequence

Suppose we call
`fib(4)`:



Example: Greatest Common Divisor

- Euclidean algorithm

E.g., gcd of 48 and 18:

$$\begin{array}{r} 2 \\ 18 \overline{) 48} \\ \underline{36} \\ 12 \end{array}$$

18 ÷ 12

$$\begin{array}{r} 1 \\ 12 \overline{) 18} \\ \underline{12} \\ 6 \end{array}$$

12 ÷ 6

$$\begin{array}{r} 2 \\ 6 \overline{) 12} \\ \underline{12} \\ 0 \end{array}$$

gcd = 6

A recursive definition

$$\text{gcd}(x, y) = \begin{cases} x, & \text{if } y = 0 \\ \text{gcd}(y, \text{remainder of } x / y), & \text{otherwise} \end{cases}$$

```
int gcd(int x, int y)
{
    if (y == 0)
        return x;
    else
        return gcd(y, x%y);
}
```

gcd.cpp

Example: Palindrome

- Recall that a palindrome is a word that reads the same forward and backward, e.g., level, noon, racecar

Recursive algorithm

r	a	c	e	c	a	r
---	---	---	---	---	---	---

To check if a string $s[0..n-1]$ is a palindrome,

1. if $n < 2$, s is a palindrome
2. otherwise, s is a palindrome if and only if $s[0]$ is the same as $s[n-1]$ and $s[1..n-2]$ is a palindrome

```
bool is_palindrome( string s )  
{  
    if (s.length() < 2)  
        return true;  
    else  
        return (s[0] == s[s.length()-1])  
            && is_palindrome(s.substr(1,s.length()-2));  
}
```

palindrome_recursive.cpp

Example: Tower of Hanoi

- The **Tower of Hanoi** is a mathematical game, consisting of three rods and disks of different sizes which can slide onto any rod.
- The puzzle starts with the disks neatly stacked in order of size on one rod, the smallest at the top, thus making a conical shape.
- Objectives: To move the entire stack to another rod.
- Rules:
 - Only one disk may be moved at a time
 - The removed disk must be placed on one of the rods
 - No disk may be placed on top of a smaller disk

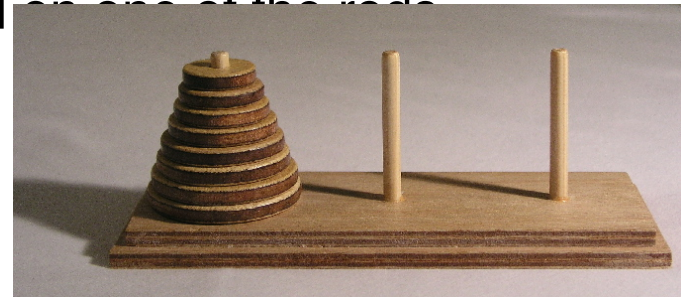
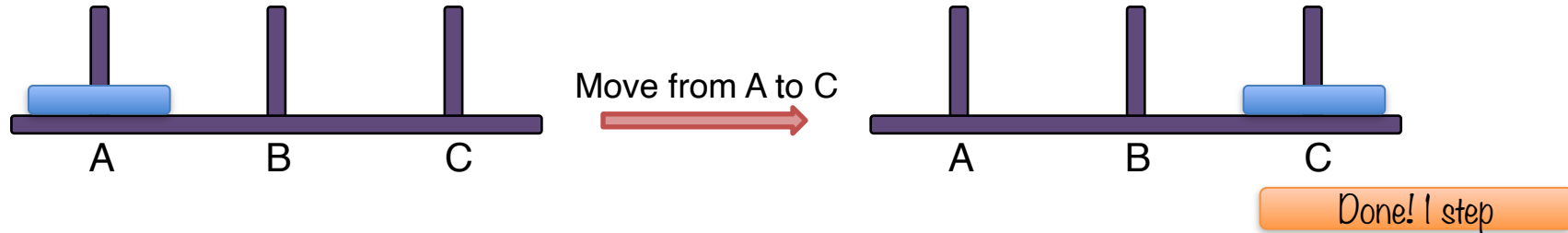


Image from Wikimedia Commons

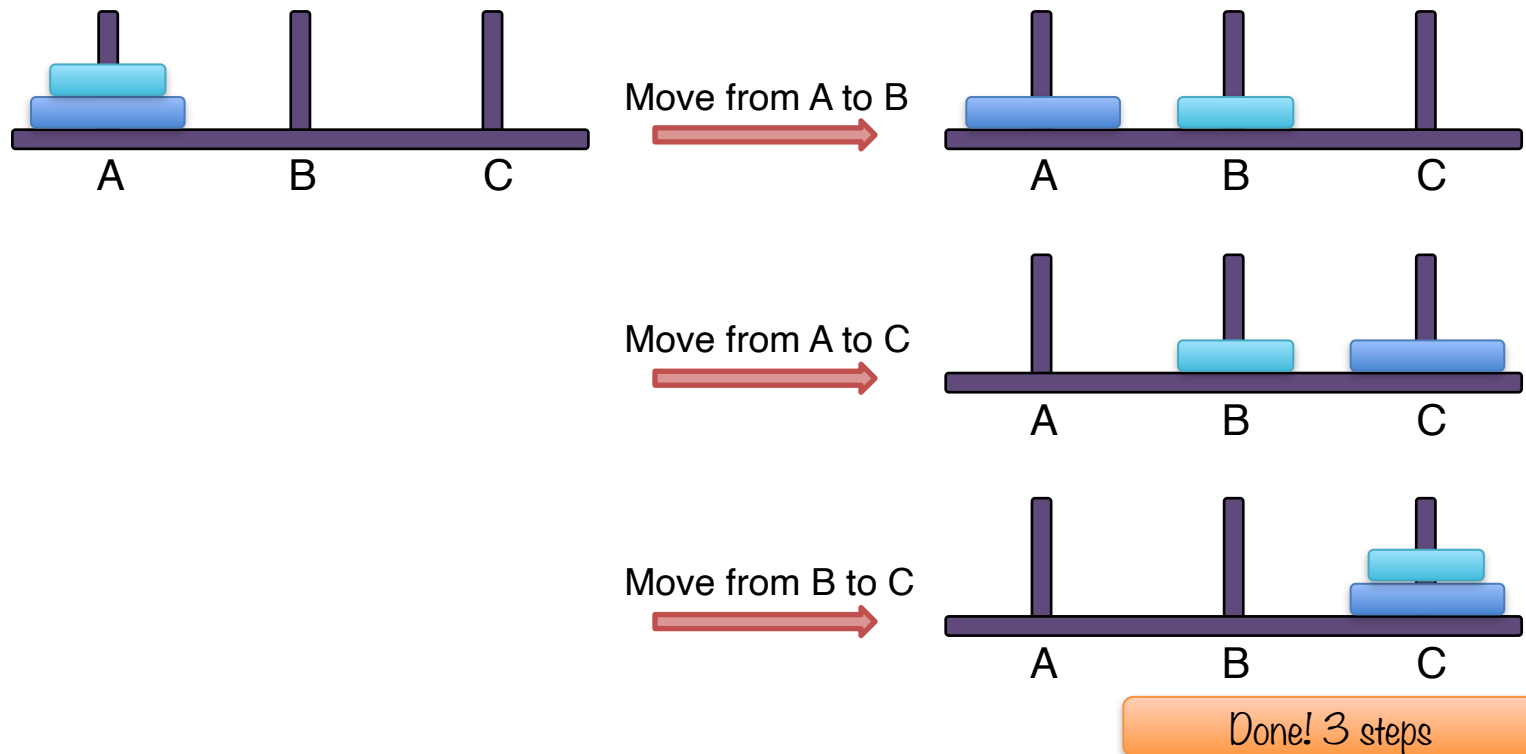
Tower of Hanoi

Suppose the task is to move the stack from rod A to rod C

What if the initial stack contains 1 disk only?

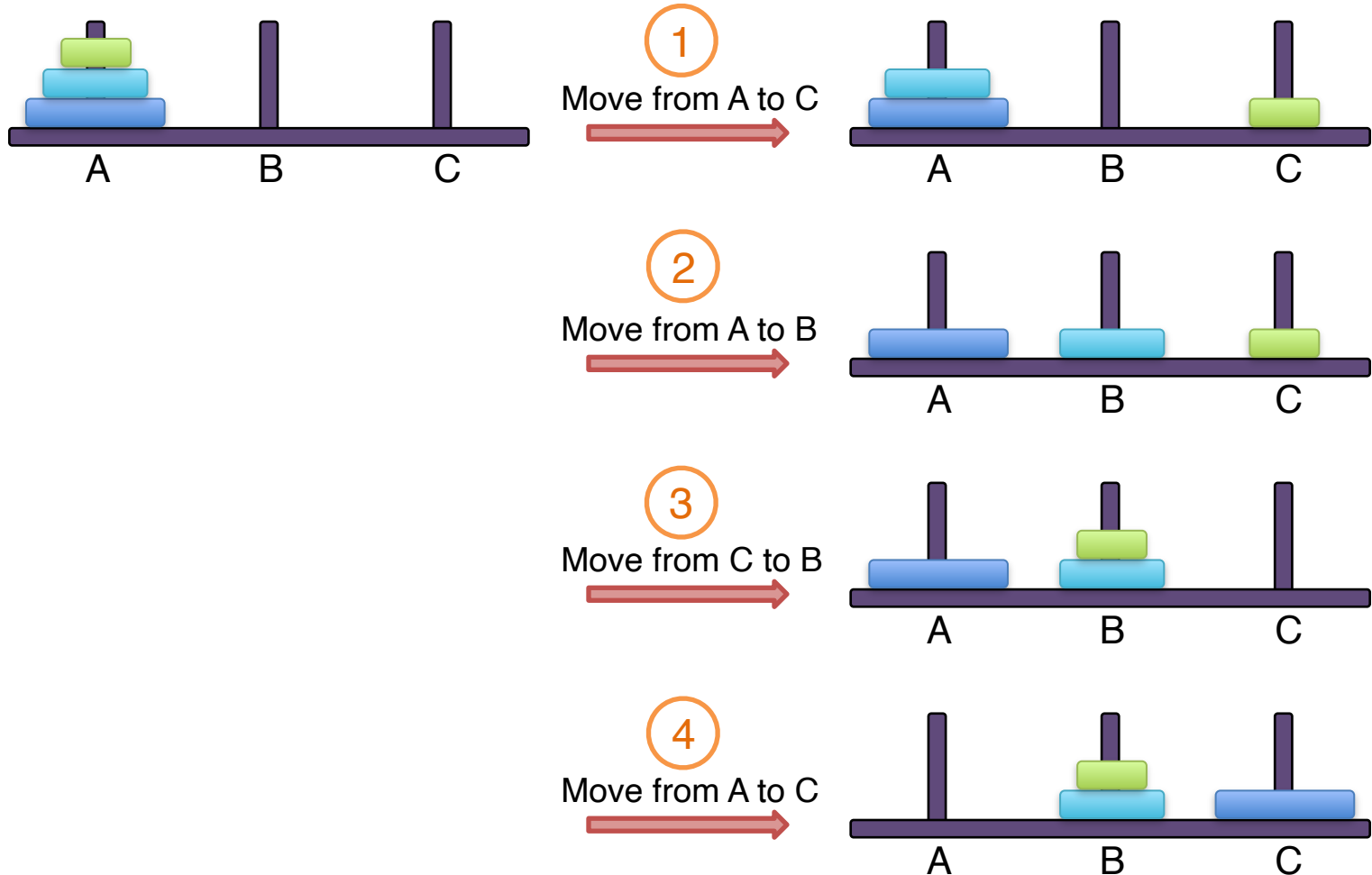


What if the initial stack contains 2 disks?

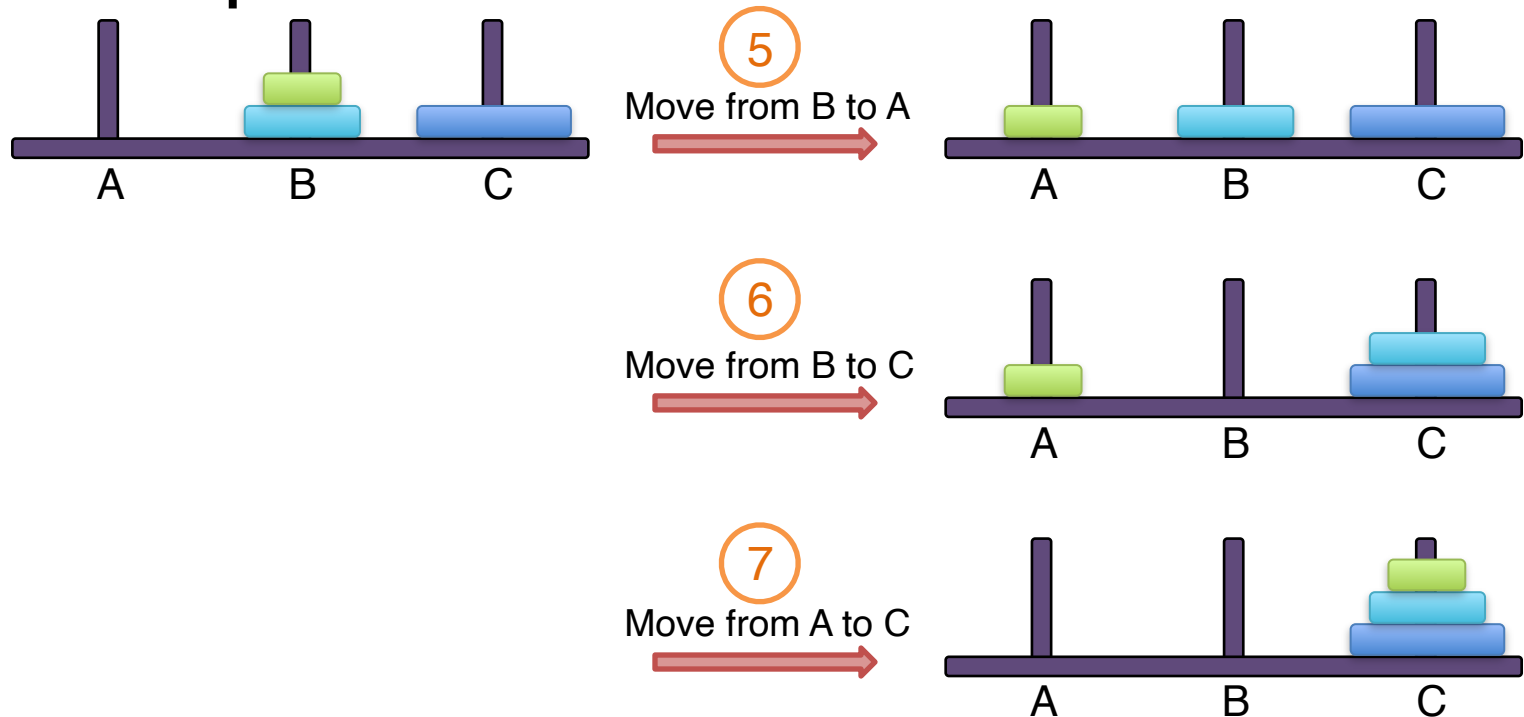


Example: Tower of Hanoi

What if the initial stack contains 3 disks?



Example: Tower of Hanoi



Done! 7 steps

What if the initial stack contains 0-4 disks???

Look at the example for moving 3 disks:
Steps 1 to 3 essentially move a stack of 2 disks from A to B

Step 4 moves a disk (the lowest of the initial stack) from A to C

Steps 5 to 7 essentially move a stack of 2 disks from B to C

A recursive algorithm!

Example: Tower of Hanoi

Recursive algorithm

To move a stack of n disks from rod A to rod C, $n \geq 1$

1. Move the top $n - 1$ disks from A to B, using C as an intermediate rod
2. Move the remaining 1 disk from A to C
3. Move the top $n - 1$ disks from B to C, using A as an intermediate

No. of disks to move

Source rod

Destination rod

Intermediate rod

```
void move(int n, char src, char des, char tmp)
{
    if (n == 1)
        cout << "Move disk from " << src << " to " << des << endl;
    else {
        move( n-1, src, tmp, des);
        move( 1, src, des, tmp);
        move( n-1, tmp, des, src);
    }
}
```

hanoi.cpp

Example: Tower of Hanoi

How many steps does it take to move 64 disks?

No. of steps to move n disks

$$\begin{aligned}T(n) &= 2 T(n - 1) + 1 \\&= 2 [2 T(n - 2) + 1] + 1 \\&= 2^2 T(n - 2) + 2 + 1 \\&= 2^2 [2 T(n - 3) + 1] + 2 + 1 \\&= 2^3 T(n - 3) + 2^2 + 2 + 1 \\&= \dots \\&= 2^{n-1} T(n - (n - 1)) + 2^{n-2} + \dots + 2^2 + 2 + 1 \\&= 2^{n-1} T(1) + 2^{n-2} + \dots + 2^2 + 2 + 1 \\&= 2^{n-1} + 2^{n-2} + \dots + 2^2 + 2 + 1 \\&= 2^n - 1\end{aligned}$$

Hence, it takes
 $2^{64} - 1 \approx 1.6 \times 10^{19}$ steps to move
64 disks

If it takes 1 second to move a disk physically by
hand, it would take
 5×10^{11} years to finish.

If a computer can generate 10^9 moves per second,
it still takes 500 years to generate all the moves!

Stack Overflow

- Each function call entails additional memory space (function call stack).
- There is always some limit to the memory size.
- If there is excessively long chain of recursive call, e.g., infinite recursion, **stack overflow error** may occur

Try the Tower of Hanoi program and see what's the largest n that will crash your machine ☺

Recursion vs Iteration

- Recursion is **NOT** absolutely necessary.
- Any task that can be accomplished using recursion can also be done in some other way **without** using recursion.
- The non-recursive version of a function typically uses a loop of some sort in place of recursion, hence often being referred to as **iterative version**.
- A recursively written function will usually **run slower** and **use more storage** than an equivalent iterative version (due to extra work in memory management for function calls (aka stack management)).
- Nonetheless, using recursion can sometimes make the job of programming easier and produce code that is easier to understand.

Recursion vs Iteration

Recursive example:

```
int binary_search(int A[], int lb, int ub, int value)
{
    if (lb > ub)
        return -1;
    else
    {
        int i = (lb + ub) / 2;
        if (A[i] == value)
            return i;
        else if (A[i] > value)
            return binary_search(A, lb, i - 1, value);
        else
            return binary_search(A, i + 1, ub, value);
    }
}
```

Iterative example:

```
int binary_search(int A[], int size, int value)
{
    int lb = 0, ub = size - 1;

    while (lb <= ub)
    {
        int i = (lb + ub) / 2;
        if (A[i] == value)
            return i;
        else if (A[i] > value)
            ub = i - 1;
        else
            lb = i + 1;
    }
    return -1;
}
```

binary_search.cpp

Tutorial Problems - Recursion

SUM OF NATURAL NUMBERS

Sum of Natural Numbers

- Write a program that calculates the sum of the first n natural numbers, i.e., $1 + 2 + \dots + n$.
- Create a new file and save it as **sum.cpp**
- Write a **main** function that
 - ask a user to input a positive integer n
 - call a function **sum(n)** to calculate the sum
 - output the result
- Write a **sum()** function (see also next slide) that
 - takes an integer n as input parameter
 - return the result of $1 + 2 + \dots + n$

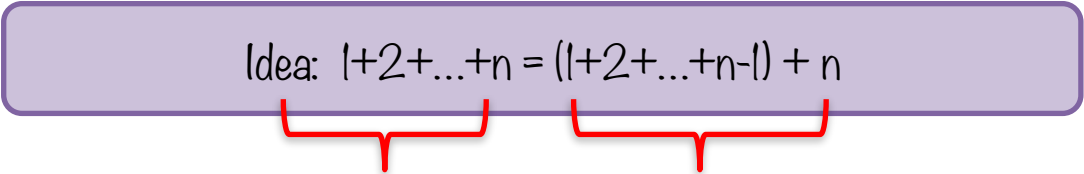
sum_complete.cpp provides the completed version of this tutorial problem. You may compile and run it to see the expected results first

Sample output (user input in orange):

```
Enter a positive integer: 5
Sum of first 5 natural numbers = 15
```


Sum of Natural Numbers

- **First version of `sum()` – iterative version**
 - Write a `sum()` function so that it makes use of a loop to calculate the sum
 - Run and test your program
- **Second version of `sum()` – recursive version**
 - Write a `sum()` function which makes use of recursion to calculate the sum


$$\text{Idea: } 1+2+\dots+n = (1+2+\dots+n-1) + n$$

This is `sum(n)`

So what is this?

- What is the base case? What is the general case?
- Run and test your program

Go to see [Hints](#) if you want the answer to these two questions

Tutorial Problems - Recursion

LARGEST ELEMENT IN AN ARRAY

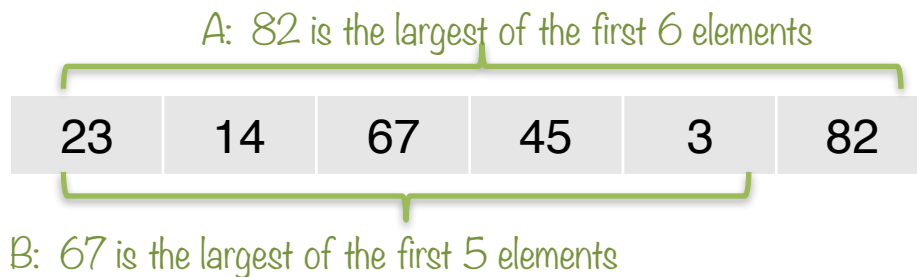
Largest Element in an Array

- Write a program to find the largest element in an array
- Open **largest_element_incomplete.cpp**
- Study the **main** function. It
 - generates a set of random positive numbers in an array
 - outputs the numbers to the screen
 - determines the largest element in the array by calling **largest()**
 - outputs the largest element

largest_element.cpp provides the complete version of this tutorial problem.

Largest Element in an Array

- Write the `largest_element()` function that uses a loop to determine the largest element in an array
 - First determine the function prototype. Look at how it is called in `main()`. What should be the input parameters? What should be the return value?
 - Finish the function body. Compile and run the program.
- Write the `largest_element()` function that uses recursion to determine the largest element in an array



How to determine A using the results of B?

- What is the base case? What is the general case?

Go to see [Hints](#) if you want the answer to these two questions

Tutorial Problems - Recursion

REVERSING A STRING

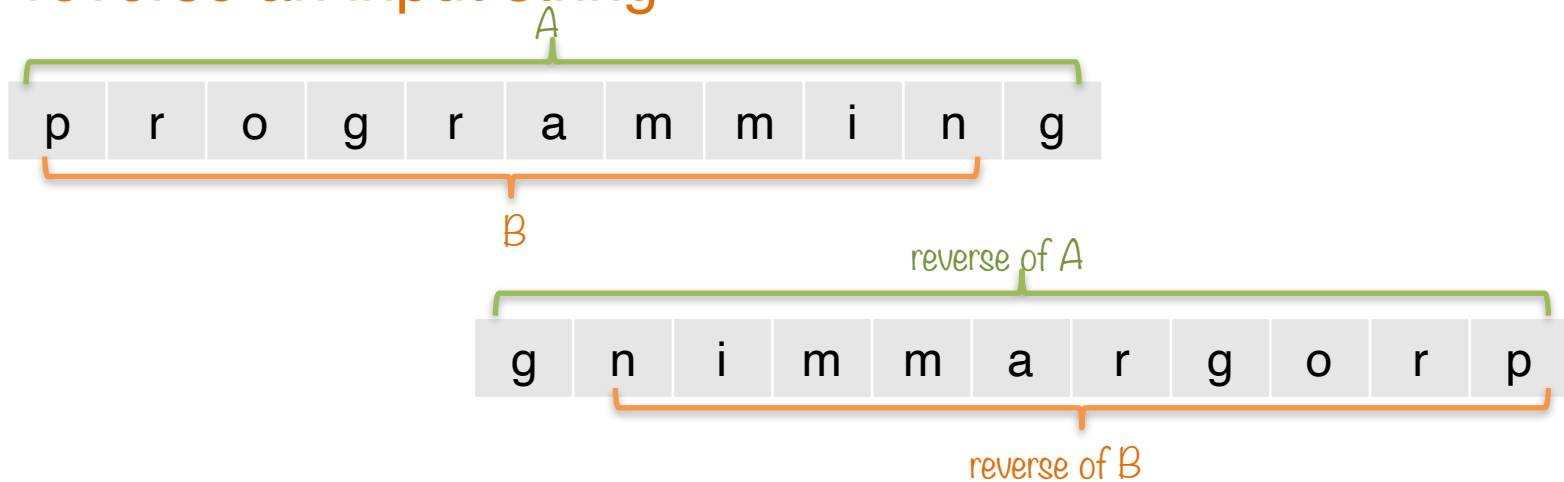
Reversing a String

- Write a program to reverse an input string.
- Open **string_reverse_incomplete.cpp**
- Study the **main** function. It
 - asks the user to input a string
 - reverse the string by calling **reverse()**
 - print out the reversed string
- Write the **reverse()** function that uses a loop to reverse an input string
 - First determine the function prototype. Look at how it is called in **main()**. What should be the input parameters? What should be the return value?
 - Finish the function body. Compile and run the program.

string_reverse.cpp provides the complete version of this tutorial problem.

Reversing a String

- Write the `reverse()` function that uses recursion to reverse an input string



- What is the base case?
What is the general case?

How is reverse of A and reverse of B related?

Go to see [Hints](#) if you want the answer to these two questions

Hints

Note that these are only suggestions.
You may come up with other solutions that
work as well!

- Sum of Natural Numbers

$$\begin{aligned} \text{sum} &= 1, && \text{if } n = 1 \\ \text{sum}(n) &= \text{sum}(n-1) + n, && \text{if } n > 1 \end{aligned}$$

- Largest Element in an Array

$$\begin{aligned} \text{largest}(\text{array}[0..n-1]) &= -1, && \text{if } n < 1 \\ \text{largest}(\text{array}[0..n-1]) &= \max(\text{largest}(\text{array}[0..n-2]), \text{array}[n-1]), && \\ \text{otherwise} &&& \end{aligned}$$

- Reverse of a String
- $$\begin{aligned} \text{reverse}(s[0..n-1]) &= s, && \text{if length of } s = 0, \\ \text{reverse}(s[0..n-1]) &= s[n-1] + \text{reverse}(s[0..n-2]), && \text{otherwise} \end{aligned}$$

PROBLEMS

Problem 1

Write a program that determines the length of a string by using for loop. DO NOT use any functions provided by <string>.

Problem 2

```
1  #include <iostream>
2  using namespace std;
3
4  void replaceAll(string &input, string from, string to);
5
6  int main() {
7      string input;
8      getline(cin, input);
9      cout << "Before replace:" << endl << input << endl;
10     replaceAll(input, "HKU", "The University of Hong Kong");
11     cout << "After replace:" << endl << input << endl;
12     replaceAll(input, "The University of Hong Kong", "H.K.U.");
13     cout << "After replace:" << endl << input << endl;
14     return 0;
15 }
```

Your task is to implement the `replaceAll` function so that it will update the string `input` by replacing all occurrences of `from` by the string `in to`. Here is a sample input (underlined) and output of the program:

I study at HKU; I love HKU!

Before replace:

I study at HKU; I love HKU!

After replace:

I study at The University of Hong Kong; I love The University of Hong Kong!

After replace:

I study at H.K.U.; I love H.K.U.!

Problem 3

Consider the following program:

```
1  #include <iostream>
2  using namespace std;
3
4  string returnNickNameGivenLastName(string name[], int n, string lastName);
5
6  int main() {
7      string name[] = { "Loretta, Choi (Loretta)",
8                        "Kin Hei, Kwok (Haley)",
9                        "Tsz Hei, Tse (Henry)",
10                       "Luv, Khanna (Luv)",
11                       "Tsz Wa, Tseng (Lucas)",
12                       "Tsz Ching, Fung (Sara)"};
13
14     int numOfUsers = 6;
15     string lastName;
16     cin >> lastName;
17     cout << returnNickNameGivenLastName(name, numOfUsers, lastName) << endl;
18     return 0;
19 }
```

Problem 3

The function `returnNickNameGivenLastName` will search in the string array `name` for the string `lastName` of a user and returns the nickname of the matched user.

Each slot of the `name[]` array stores the name of a user. It uses the following format:

[First name][comma][space][Last name][space][open bracket][Nickname][close bracket]

Note that the second input parameter of the function is an integer denoting the number of users (i.e., number of slots in the `name[]` array).

The function returns a `string`, which is the nickname of the matched user (you may assume that there is at most one match).

The function returns the string "Not found!" if no match is found. Here is a sample input (underlined) and output of the program:

Choi
Loretta

Optional.

For those who would like to challenge yourselves.

Even for those of you who are beginners in C++ programming, it's highly recommended for you to take a look at these problems and try to tackle them as well.

You are welcome to discuss these problems in the Moodle forum.

CHALLENGES

Challenge 1

Write a little library system for storing and searching books with using a text file. The data should include name, author, call number and subject. No restriction on output format.

You may start with the following struct definition, function prototypes and main function design:

Challenge 1

```
1  struct Book{
2      string name;
3      string author;
4      int call_number;
5      string subject;
6  };
7  typedef struct Book Book;
8
9  bool equal(const Book a, const Book b);
10 void insert(const Book book);
11 void search(const Book book);
12
13 int main(){
14     string command;
15
16     while(cin >> command){
17         if(command == "Input"){
18             Book book
19             cin << book.name << book.author << book.call_number << book.subject;
20
21             input(book);
22         }
23         if(command == "Search"){
24             Book book
25             cin << book.name << book.author << book.call_number << book.subject;
26
27             search(book);
28         }
29         if(command == "Exit"){
30             return 0;
31         }
32     }
33 }
```


Challenge 2

This question is on file I/O and string manipulation.

Write a program to read a file and report the number of occurrences of each word in the file. Your program will not distinguish between upper- and lower-case words. E.g., “hello”, “Hello”, “HELLO” are the same. The words must be output in lowercase and sorted in ascending order of their frequencies. Words with same frequency are ordered lexicographically.

Example:

Contents of input file:

```
Hello, where are you going to?  
Are you going to school to ... say hello?
```

Your program output:

```
say 1  
school 1  
where 1  
are 2  
going 2  
hello 2  
you 2  
to 3
```

Hint: By including `<cctype>` you can use the function `ispunct()` to test whether a character is a punctuation or not.