Section Overview

Characters and Strings

- Character functions
- C-style Strings
- Working with C-style Strings
- C++ Strings
- Working with C++ Strings

Character Functions

```
#include <cctype>
```

```
#include <cctype>

function_name(char)
```

- Functions for testing characters
- Functions for converting character case

Character Functions

Testing characters

`isalpha(c)`	True if c is a letter
`isalnum(c)`	True if c is a letter or digit
`isdigit(c)`	True if c is a digit
`islower(c)`	True if c is lowercase letter
`isprint(c)`	True if c is a printable character
`ispunct(c)`	True if c is a punctuation character
`isupper(c)`	True if c is an uppercase letter
`isspace(c)`	True if c is whitespace

Character Functions

Converting characters

`tolower(c)`	returns lowercase of c
`toupper(c)`	returns uppercase of c

C-style Strings

- Sequence of characters
  - contiguous in memory
  - implemented as an array of characters
  - terminated by a null character (null)
    - null – character with a value of zero
  - Referred to as zero or null terminated strings

- String literal
  - sequence of characters in double quotes, e.g. “Frank”
  - constant
  - terminated with null character

C-style Strings

"C++ is fun"

C	+	+		i	s		f	u	n	\0

C-style Strings

declaring variables

```
char my_name[] {"Frank"};
```

F	r	a	n	k	\0

```
my_name[5] = 'y'; // Problem
```

C-style Strings

declaring variables

```
char my_name[8] {"Frank"};
```

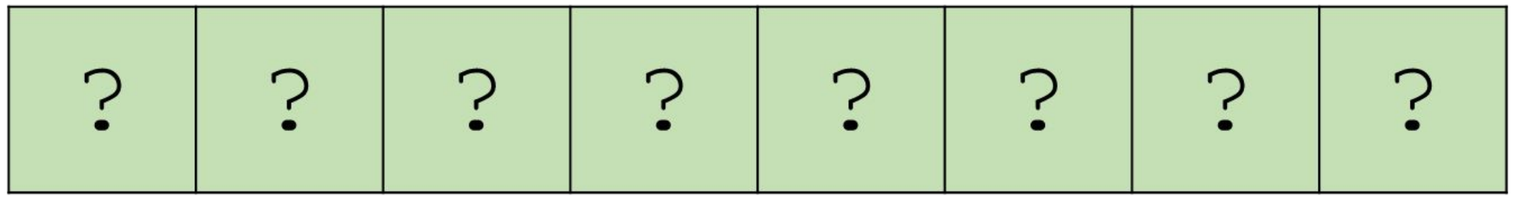
F	r	a	n	k	\0	\0	\0

```
my_name[5] = 'y'; // OK
```

C-style Strings

declaring variables

```
char my_name[8];
```




```
my_name = "Frank";              // Error

strcpy(my_name, "Frank"); // OK
```

#include <cstring>

Functions that work with C-style Strings

- Copying
- Concatenation
- Comparison
- Searching
- and others

#include <cstring>

A few examples

```
char str[80];

strcpy(str, "Hello "); // copy

strcat(str, "there");  // concatenate

cout << strlen(str);   // 11

strcmp(str, "Another"); // > 0
```

#include <cstdlib>

General purpose functions

- Includes functions to convert C-style Strings to
  - integer
  - float
  - long
  - etc.

C++ Strings

- std::string is a Class in the Standard Template Library
  - #include <string>
  - std namespace
  - contiguous in memory
  - dynamic size
  - work with input and output streams
  - lots of useful member functions
  - our familiar operators can be used (+, = , < , <=, >, >=, +=, ==, !=, []...)
  - can be easily converted to C-style Strings if needed
  - safer

C++ Strings

Declaring and initializing

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

string s1;                  // Empty
string s2 {"Frank"};        // Frank
string s3 {s2};             // Frank
string s4 {"Frank", 3};     // Fra
string s5 {s3, 0, 2};       // Fr
string s6 (3, 'X');         // XXX
```

C++ Strings

Assignment =

```
string s1;
s1 = "C++ Rocks!";

string s2 {"Hello"};
s2 = s1;
```

C++ Strings

Concatenation

```
string part1 {"C++"};
string part2 {"is a powerful"};

string sentence;
sentence = part1 + " " + part2 + " language";
        // C++ is a powerful language

sentence = "C++" + " is powerful"; // Illegal
```

C++ Strings

Accessing characters [] and at() method

```
string s1;
string s2 {"Frank"};

cout << s2[0] << endl;     // F
cout << s2.at(0) << endl; // F

s2[0] = 'f';       // frank
s2.at(0) = 'p';   // prank
```

C++ Strings

Accessing characters [] and at() method

```
string s1 {"Frank"};

for (char c: s1)
    cout << c << endl;
```

```
F
r
a
n
k
```

C++ Strings

Accessing characters [] and at() method

```
string s1 {"Frank"};

for (int c: s1)
     cout << c << endl;
```

```
70    // F
114   // r
97    // a
110   // n
107   // k
0     // null character
```

C++ Strings

Comparing

`==` `!=` `>` `>=` `<` `<=`

The objects are compared character by character lexically.

Can compare:

- two `std::string` objects
- `std::string` object and C-syle string literal
- `std::string` object and C-style string variable

C++ Strings

Comparing

```
string s1 {"Apple"};
string s2 {"Banana"};
string s3 {"Kiwi"};
string s4 {"apple"};
string s5 {s1};              // Apple

s1 == s5                     // True
s1 == s2                     // False
s1 != s2                     // True
s1 < s2                      // True
s2 > s1                      // True
s4 < s5                      // False
s1 == "Apple";               // True
```

C++ Strings

Substrings – `substr()`

Extracts a substring from a std::string

```
object.substr(start_index, length)
```

```
string s1 {"This is a test"};

cout << s1.substr(0,4); // This
cout << s1.substr(5,2); // is
cout << s1.substr(10,4); // test
```

C++ Strings

Searching – `find()`

Returns the index of a substring in a std::string

```
object.find(search_string)
```

```
string s1 {"This is a test"};

cout << s1.find("This");  // 0
cout << s1.find("is");  // 2
cout << s1.find("test");  // 10
cout << s1.find('e');   // 11
cout << s1.find("is", 4);   // 5
cout << s1.find("XX");  // string::npos
```

C++ Strings

Removing characters – `erase()` and `clear()`

Removes a substring of characters from a std::string

```
object.erase(start_index, length)
```

```
string s1 {"This is a test"};

cout << s1.erase(0,5);  // is a test
cout << s1.erase(5,4);  // is a
s1.clear();              // empties string s1
```

C++ Strings

Other useful methods

```
string s1 {"Frank"};

cout << s1.length() << endl; // 5

s1 += " James";
cout << s1 << endl;     // Frank James
```

Many more...

C++ Strings

Input >> and `getline()`

Reading std::string from cin

```
string s1;
cin >> s1;            // Hello there
              // Only accepts up to the first space
cout << s1 << endl;   // Hello

getline(cin, s1);  // Read entire line until \n
cout << s1 << end;    // Hello there

getline(cin, s1, 'x');  // this isx
cout << s1 << endl;  // this is
```

Functions

- Function
  - definition
  - prototype
  - Parameters and pass-by-value
  - `return` statement
  - default parameter values
  - overloading
  - passing arrays to function
  - pass-by-reference
  - `inline` functions
  - `auto` return type
  - recursive functions

What is a function?

- C++ programs
  - C++ Standard Libraries (functions and classes)
  - Third-party libraries (functions and classes)
  - Our own functions and classes

- Functions allow the modularization of a program
  - Separate code into logical self-contained units
  - These units can be reused

What is a function?

```
int main() {

 // read input
   statement1;
   statement2;
   statement3;
   statement4;

 // process input
   statement5;
   statement6;
   statement7;

 // provide output
   statement8;
   statement9;
   statement10;

   return 0;
}
```

Modularized Code

```
int main() {

 // read input
   read_input();

 // process input
   process_input();

 // provide output
   provide_output();

   return 0;
}
```

What is a function?

```
int main() {

    read_input();

    process_input();

    provide_output();

    return 0;
}
```

```
read_input() {
    statement1;
    statement2;
    statement3;
    statement4;
}

process_input() {
    statement5;
    statement6;
    statement7;
}

provide_output() {
    statement8;
    statement9;
    statement10;
}
```

What is a function?

Boss/Worker analogy

- Write your code to the function specification
- Understand what the function does
- Understand what information the function needs
- Understand what the function returns
- Understand any errors the function may produce
- Understand any performance constraints

- Don’t worry about HOW the function works internally
  - Unless you are the one writing the function!

What is a function?

Example `<cmath>`

- Common mathematical calculations
- Global functions called as:

```
function_name(argument);
function_name(argument1, argument2, …);
```

```
cout << sqrt(400.0) << endl;  // 20.0
double result;
result = pow(2.0, 3.0);      // 2.0^3.0
```

What is a function?

User-defined functions

- We can define our own functions
- Here is a preview

```
/* This is a function that expects two integers a and b
   It calculates the sum of a and b and returns it to the caller
   Note that we specify that the function returns an int
*/

int add_numbers(int a, int b)
{
    return a + b;
}

// I can call the function and use the value that is returns

cout << add_numbers(20, 40);
```

What is a function?

User-defined functions

- Return zero if any of the arguments are negative

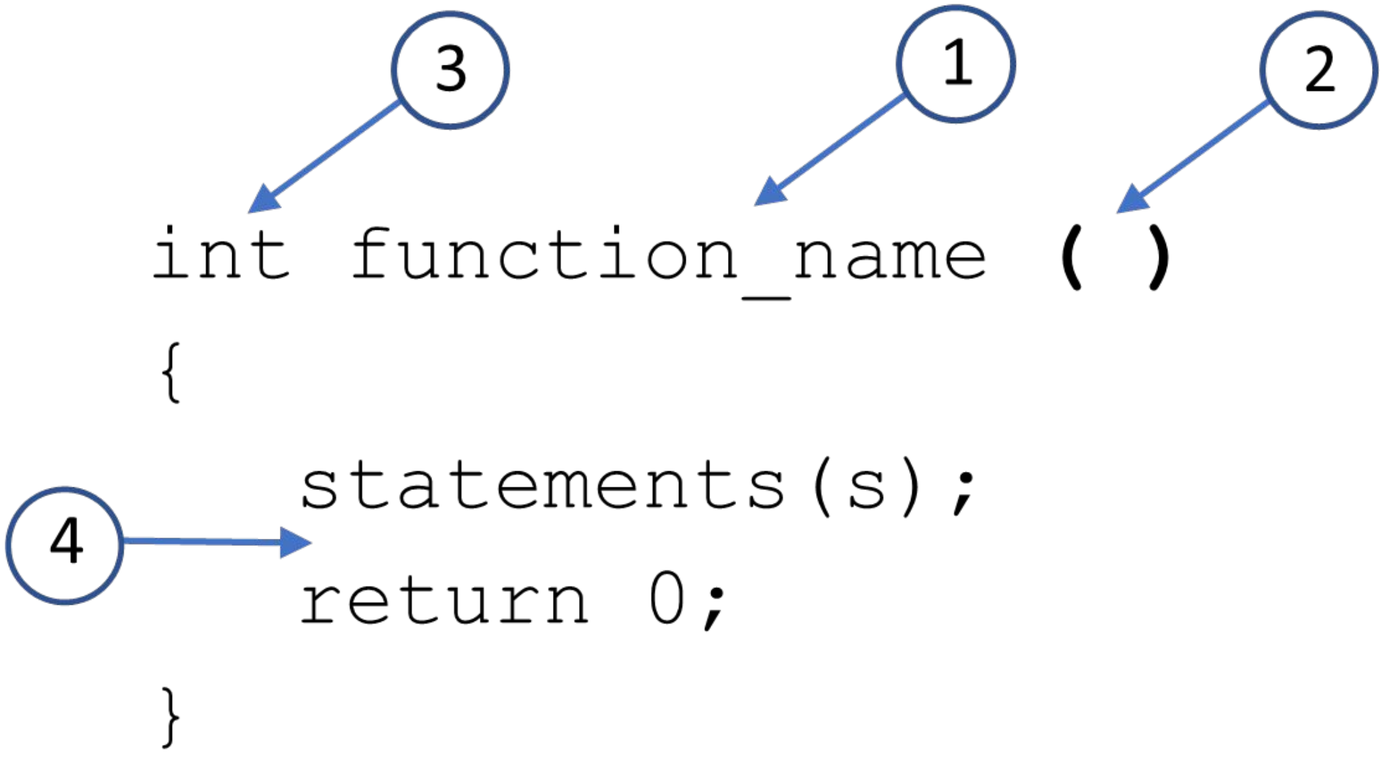
```
/* This is a function that expects two integers a and b
   It calculates the sum of a and b and returns it to the caller
   Only if a or b are non-negative. Otherwise, it returns 0
   Note that we specify that the function returns an int
*/

int add_numbers(int a, int b)
{
    if (a < 0 || b < 0)
        return 0;
    else
        return a + b;
}
```

Defining Functions

- name
  - the name of the function
  - same rules as for variables
  - should be meaningful
  - usually a verb or verb phrase
- parameter list
  - the variables passed into the function
  - their types must be specified
- return type
  - the type of the data that is returned from the function
- body
  - the statements that are executed when the function is called
  - in curly braces {}

Defining Functions

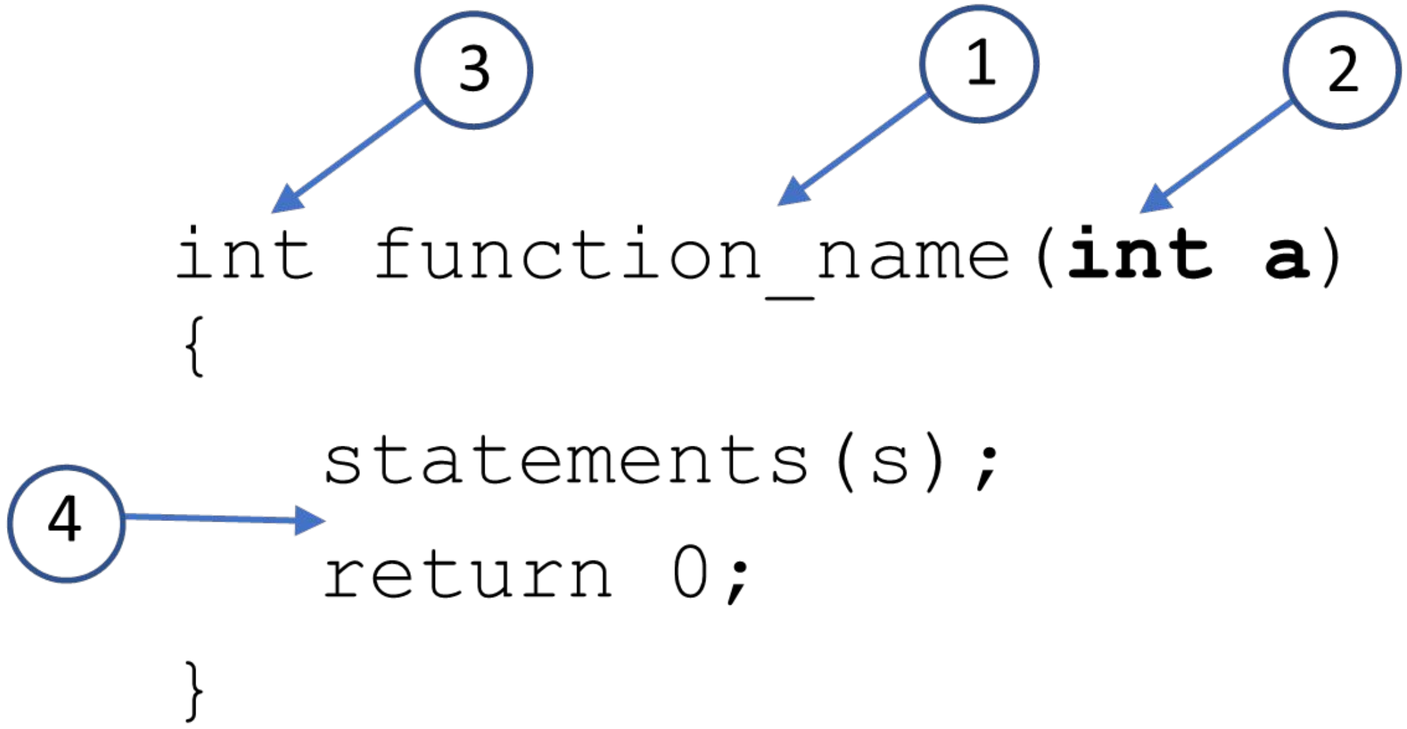
Example with no parameters



1. Name
2. Parameters
3. Return type
4. Body

Defining Functions

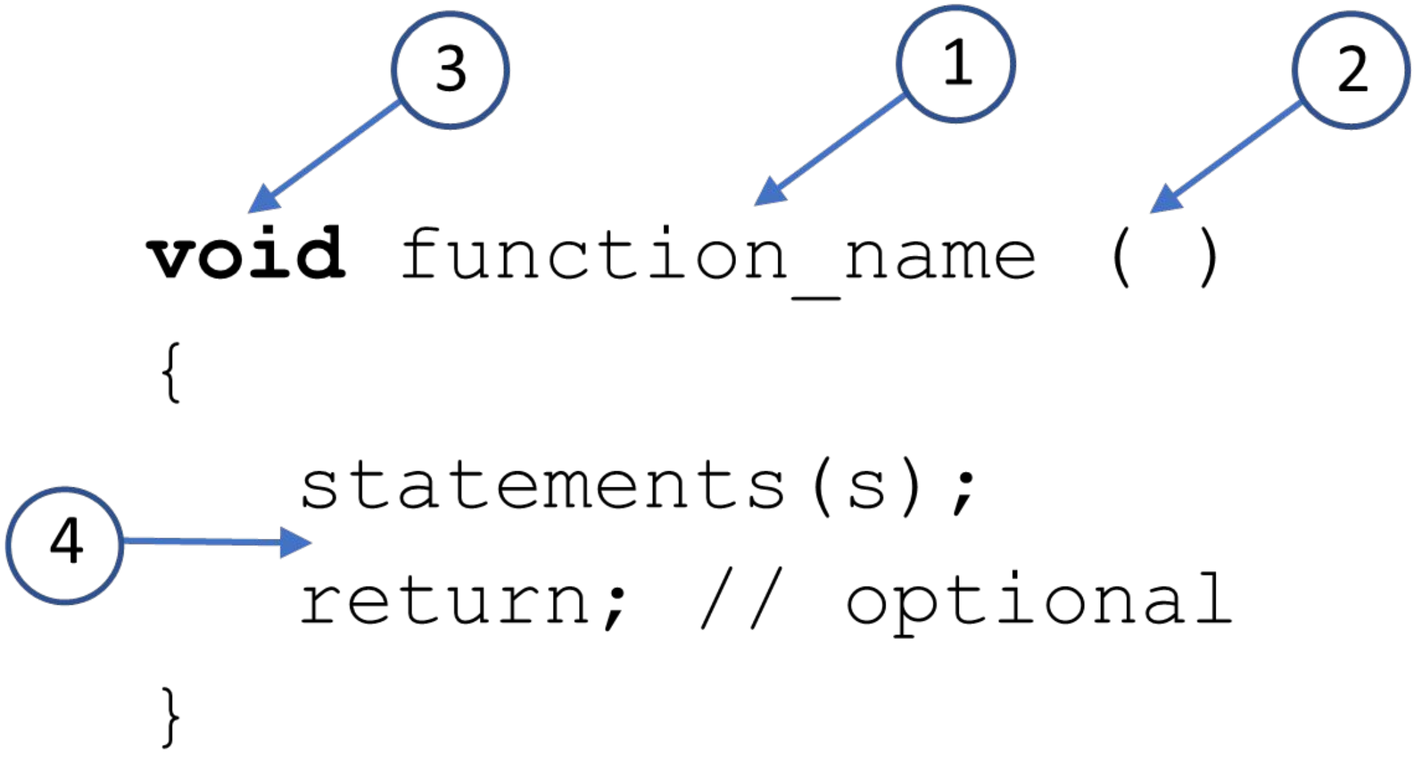
Example with 1 parameter



1. Name
2. Parameters
3. Return type
4. Body

Defining Functions

Example with no return type (void)



1. Name
2. Parameters
3. No return type
4. Body

Defining Functions

Example with multiple parameters

```
void function_name(int a, std::string b)
{
    statements(s);
    return; // optional
}
```

Defining Functions

A function with no return type and no parameters

```
void say_hello () {
    cout << "Hello" << endl;
}
```

Calling a function

```
void say_hello () {
    cout << "Hello" << endl;
}

int main() {
    say_hello();
    return 0;
}
```

Calling a function

```
void say_hello () {
    cout << "Hello" << endl;
}

int main() {
    for (int i{1} i<=10; ++i)
        say_hello();
    return 0;
}
```

Calling a function

```
void say_world () {
    cout << " World" << endl;
}

void say_hello () {
    cout << "Hello" << endl;
    say_world();
}

int main() {
    say_hello();
    return 0;
}
```

Calling a function

```
void say_world () {
    cout << " World" << endl;
    cout << " Bye from say_world" << endl;
}

void say_hello () {
    cout << "Hello" << endl;
    say_world();
    cout << " Bye from say_hello" << endl;
}

int main() {
    say_hello();
    cout << " Bye from main" << endl;
    return 0;
}
```

```
Hello
World
Bye from say_world
Bye from say_hello
Bye from main
```

Calling functions

- Functions can call other functions
- Compiler must know the function details **BEFORE** it is called!

```
int main() {
    say_hello();  // called BEFORE it is defined ERROR
    return 0;
}
```

```
void say_hello ()
{
    cout << "Hello" << endl;
}
```

Function Prototypes

- **The compiler must ‘know’ about a function before it is used**
  - Define functions before calling them
    - OK for small programs
    - Not a practical solution for larger programs
  - Use function prototypes
    - Tells the compiler what it needs to know without a full function definition
    - Also called forward declarations
    - Placed at the beginning of the program
    - Also used in our own header files (.h) – more about this later

Example

```
int function_name();  // prototype


int function_name()
{
    statements(s);
    return 0;
}
```

Example

```
int function_name(int);   // prototype
                          // or
int function_name(int a); // prototype


int function_name(int a) {
    statements(s);
    return 0;
}
```

Example

```
void function_name(); // prototype


void function_name()
{
    statements(s);
    return; // optional
}
```

Example

```
void function_name(int a, std::string b);
// or
void function_name(int, std::string);



void function_name(int a, std::string b)
{
    statements(s);
    return; // optional
}
```

A function with no return type and no parameters

```
void say_hello();

void say_hello() {
    cout << "Hello" << endl;
}
```

Calling a function

```
void say_hello();

int main() {
    say_hello();          // OK
    say_hello(100);         // Error
    cout << say_hello(); // Error
                         // No return value
    return 0;
}
```

Example

```
void say_hello(); // prototype
void say_world(); // prototype

int main() {
    say_hello();
    cout << " Bye from main" << endl;
    return 0;
}
```

```
void say_world () {
    cout << " World" << endl;
    cout << " Bye from say_world‟ << endl;
}

void say_hello () {
    cout << "Hello" << endl;
    say_world();
    cout << " Bye from say_hello" << endl;
}
```

Function Parameters

- When we call a function we can pass in data to that function
- In the function call they are called arguments
- In the function definition they are called parameters
- They must match in number, order, and in type

Example

```
int add_numbers(int, int);          // prototype

int main() {
    int result {0};
    result = add_numbers(100,200);  // call
    return 0;
}

int add_numbers(int a, int b) { // definition
    return a + b;
}
```

Example

```
void say_hello(std::string name) {
    cout << "Hello " << name << endl;
}

say_hello("Frank");
```

```
std::string my_dog {"Buster"};
say_hello(my_dog);
```