Week 12

NWEN 241
Systems Programming

Alvin C. Valera

alvin.valera@ecs.vuw.ac.nz

Content

- More on socket programming

- Summary

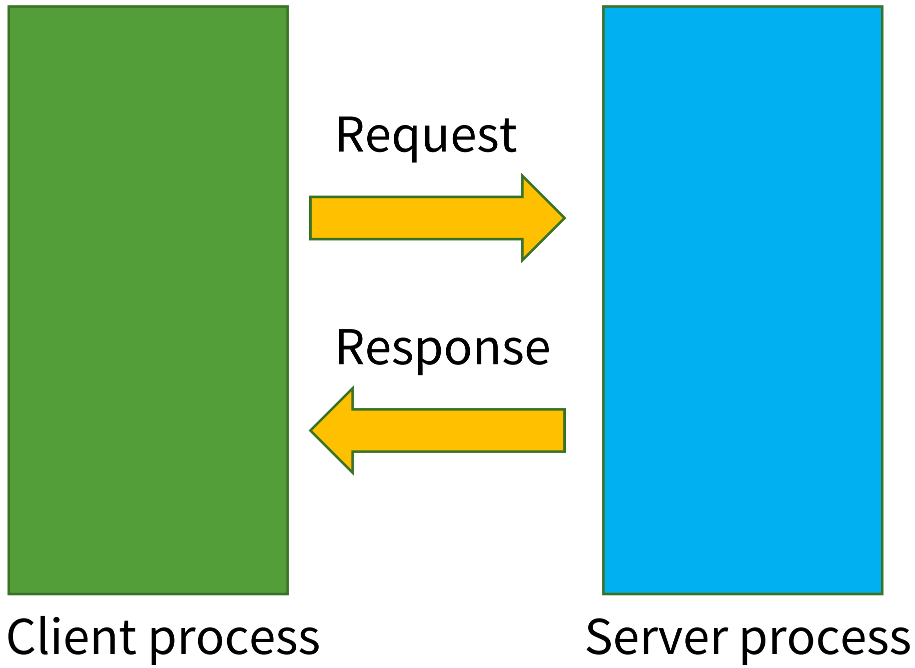
- Final exam

Recap: Client-server model

- Based on the producer-consumer model of process cooperation

- Client makes the request for some resource or service to the server process

- Server process handles the request and sends the response (result) back to the client



C and C++

Recap: Server overview

1) Create a socket with the `socket()` system call
2) Bind the socket to an address using the `bind()` system call
3) Listen for connections with the `listen()` system call
4) Accept a connection with the `accept()` system call
5) Send and receive data

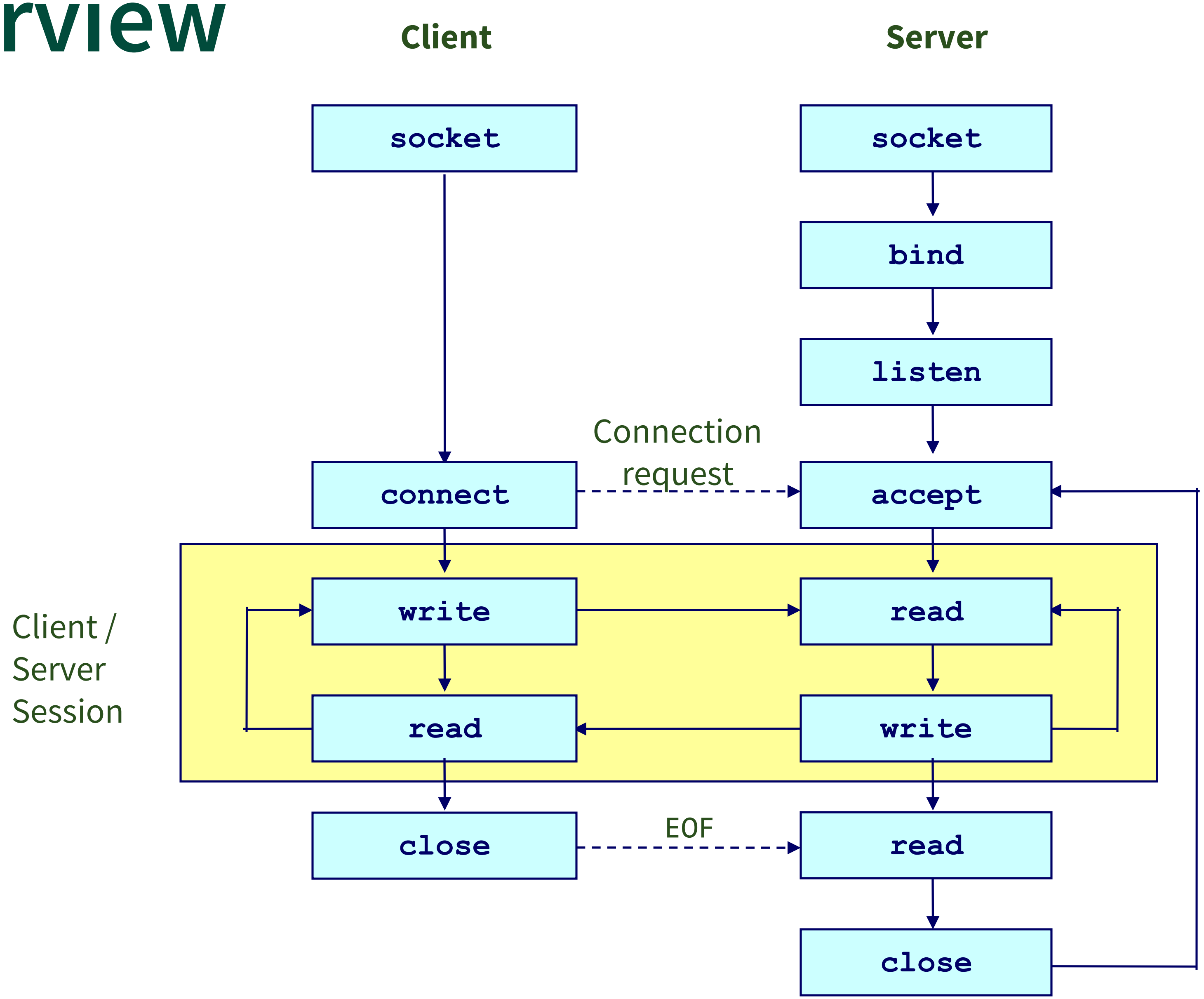
C and C++

Recap: Client overview

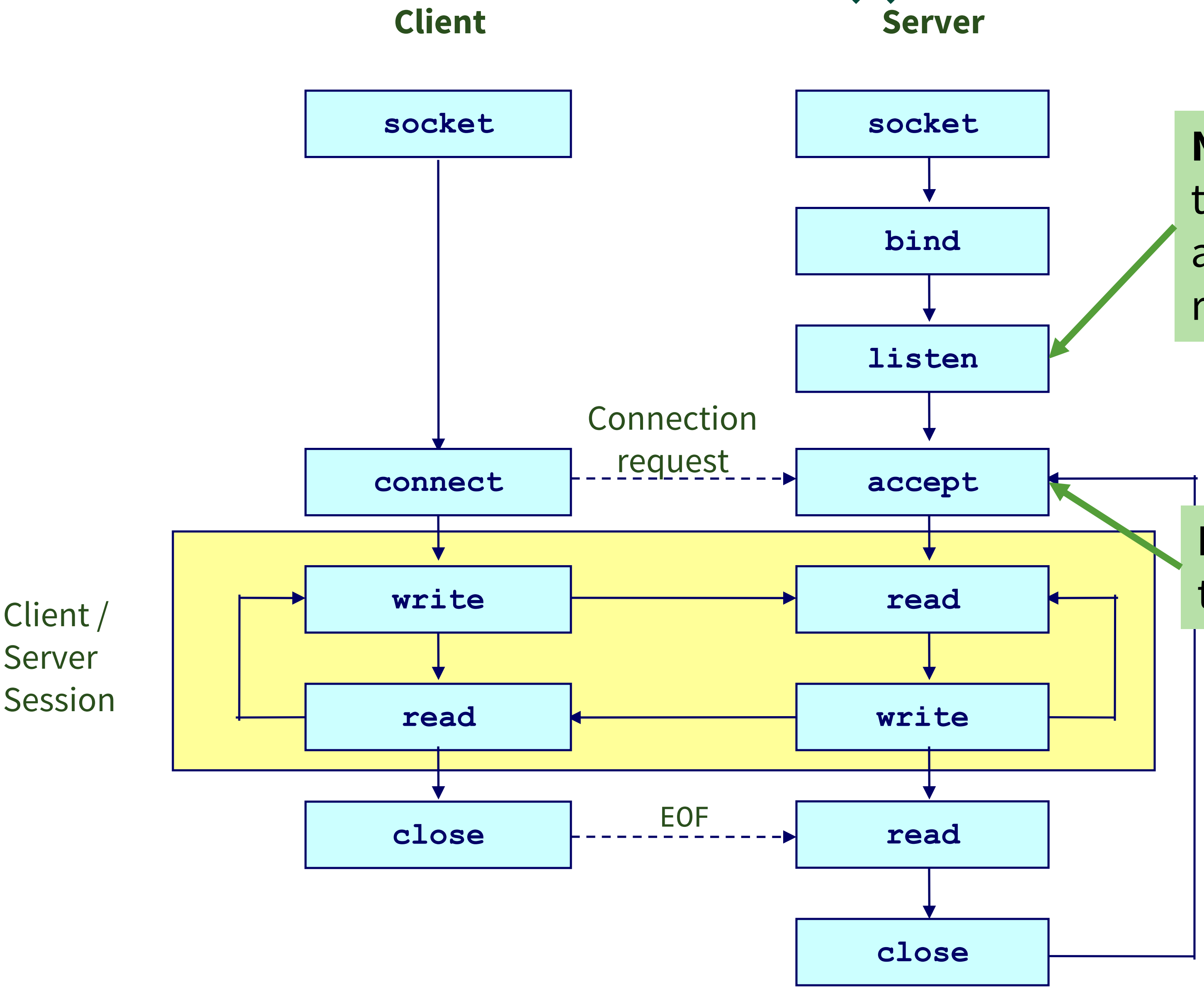
1) Create a socket with the `socket()` system call
2) Connect the socket to the address of the server using the `connect()` system call
3) Send and receive data

C and C++

Recap: Client-server communication overview



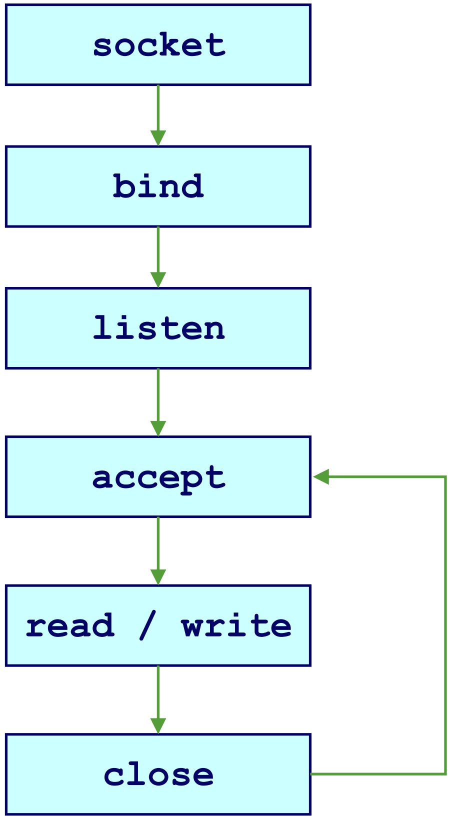
Clarification on `listen()` and `accept()`



Non-blocking call: tells OS that process is interested to accept connections and returns immediately

Blocking call: will wait until there is incoming connection

Drawback of our server



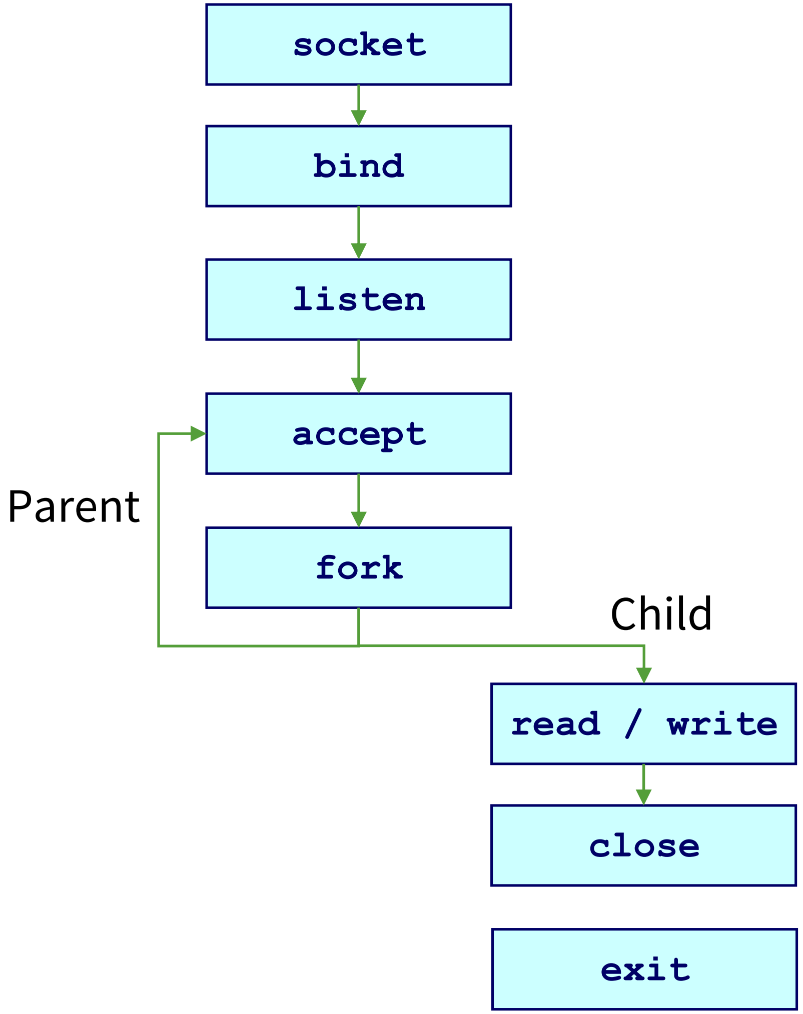
- Can only establish at most one client session at any given time

- Other connecting clients will have to wait
  - If backlog is reached, these clients may get dropped

- How to make server be able to handle more than one client session?

Socket programming with fork()

- Call fork() after accepting a client connection
- In parent process, go back to accept()
- In child process, handle communication with client



NWEN 241 Summary

- Systems programming
- Key concepts in C/C++ programming
- Final exam

Systems Programming

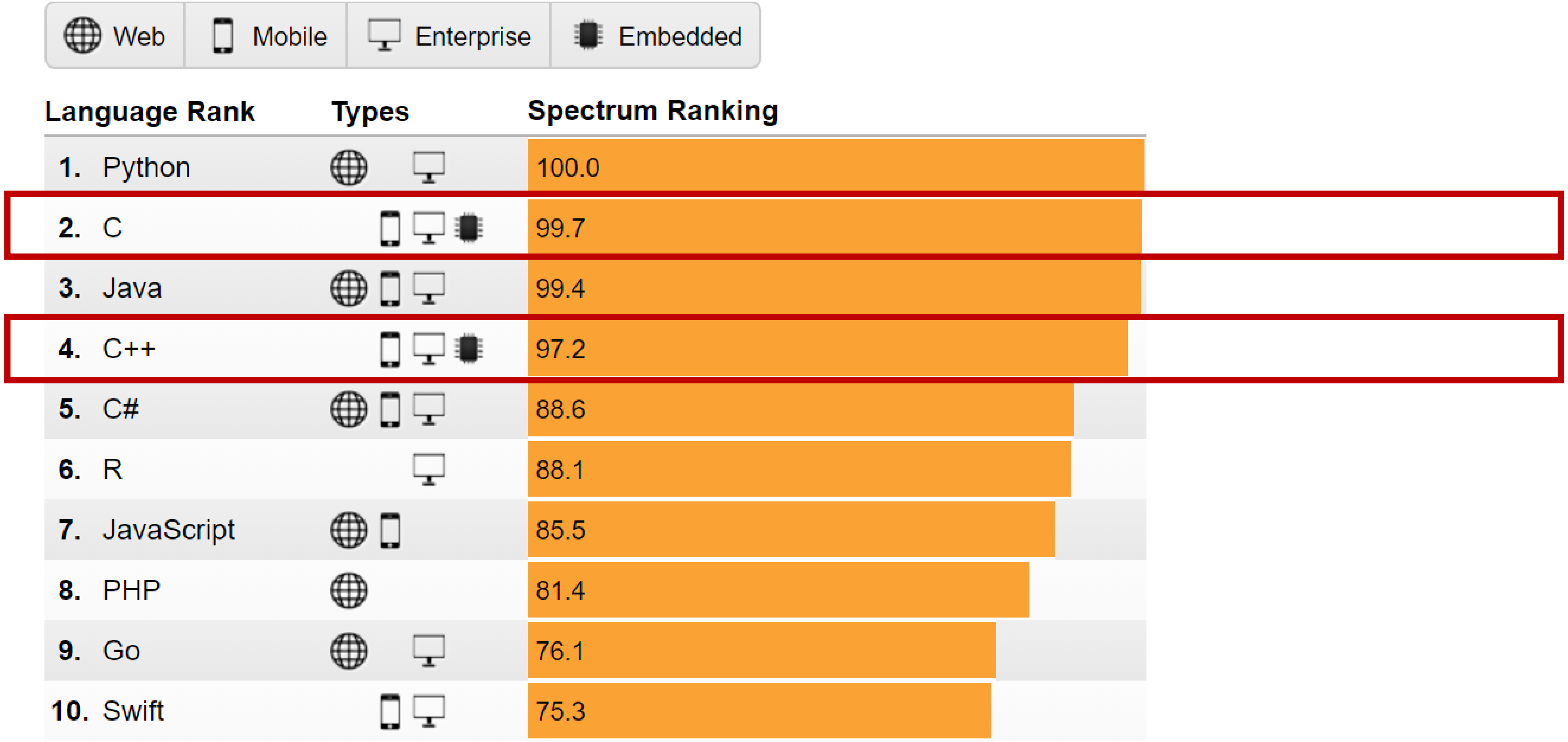
Systems programming: implementation of **systems programs** or **software**

- Systems program / software:
  - Programs that support the **operation** and **use** of the computer system itself
  - Maybe used to support other software and application programs
  - May contain **low-level** or **architecture-dependent** code

Why C and C++ for systems programming?

- C/C++ supports both **high-level** abstractions and **low-level** access to hardware at the same time
- High-level abstractions:
  - Functions
  - User-defined types (structures and classes)
  - Data structures (stacks, queues, lists)
- Low-level access to hardware:
  - Possible access to registers
  - Dynamic memory allocation
  - Inclusion of assembly code

Why learn C and C++?



🌐 Web | 📱 Mobile | 🖥 Enterprise | 🔲 Embedded

Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥	100.0
2. C	📱 🖥 🔲	99.7
3. Java	🌐 📱 🖥	99.4
4. C++	📱 🖥 🔲	97.2
5. C#	🌐 📱 🖥	88.6
6. R	🖥	88.1
7. JavaScript	🌐 📱	85.5
8. PHP	🌐	81.4
9. Go	🌐 🖥	76.1
10. Swift	📱 🖥	75.3

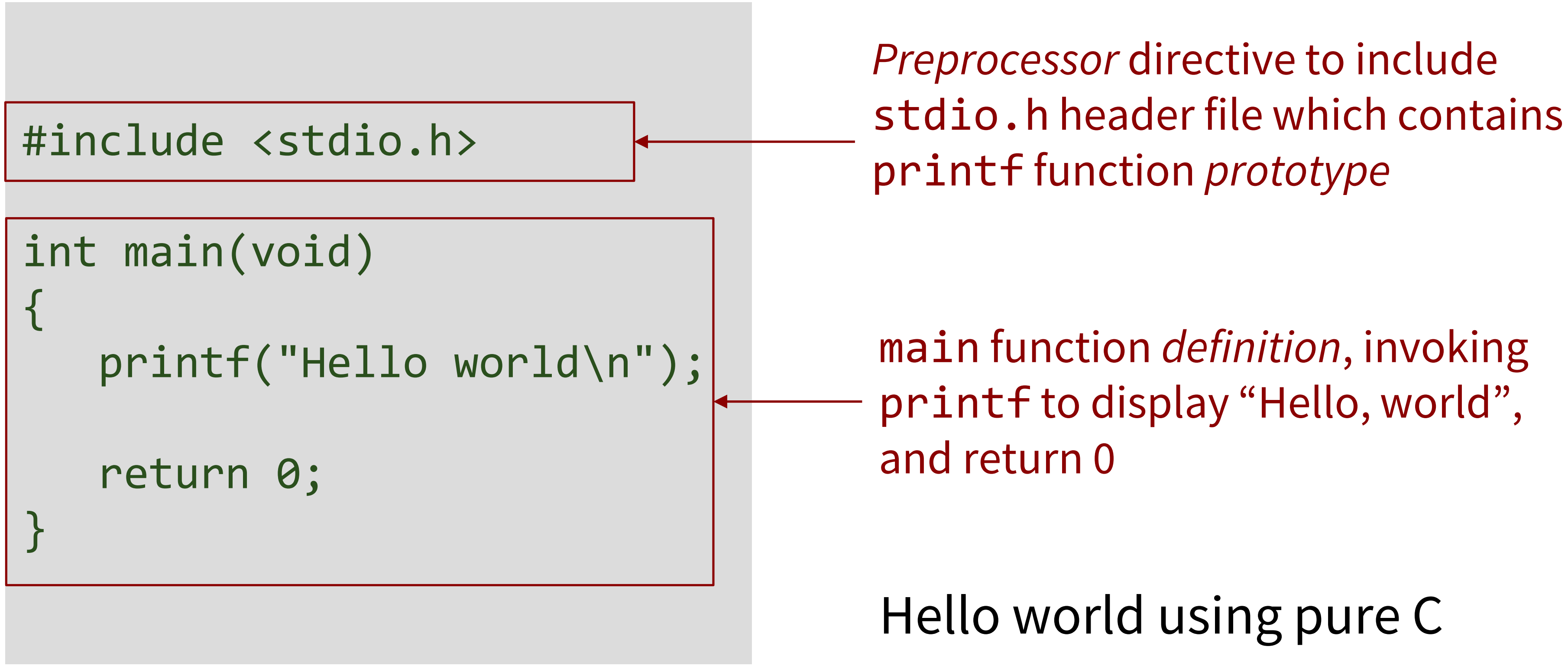
Source: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017

Key Concepts in C/C++

- Functions
- Arrays and pointers
- Dynamic memory allocation
- Structures and classes

Program Structure

- A typical C/C++ program consists of
  - 1 or more **header** files
  - 1 or more C/C++ **source** files



```
#include <stdio.h>

int main(void)
{
   printf("Hello world\n");

   return 0;
}
```

Preprocessor directive to include `stdio.h` header file which contains `printf` function *prototype*

`main` function *definition*, invoking `printf` to display “Hello, world”, and return 0

Hello world using pure C

Main Function

- A C/C++ program must have exactly one `main` function
- Execution begins with the `main` function

Does not accept command-line arguments

```
#include <stdio.h>

int main(void)
{
    …
}
```

Accepts command-line arguments

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    …
}
```

Arrays

- An array is a collection of data that holds a **fixed** number of data (values) of the **same type**

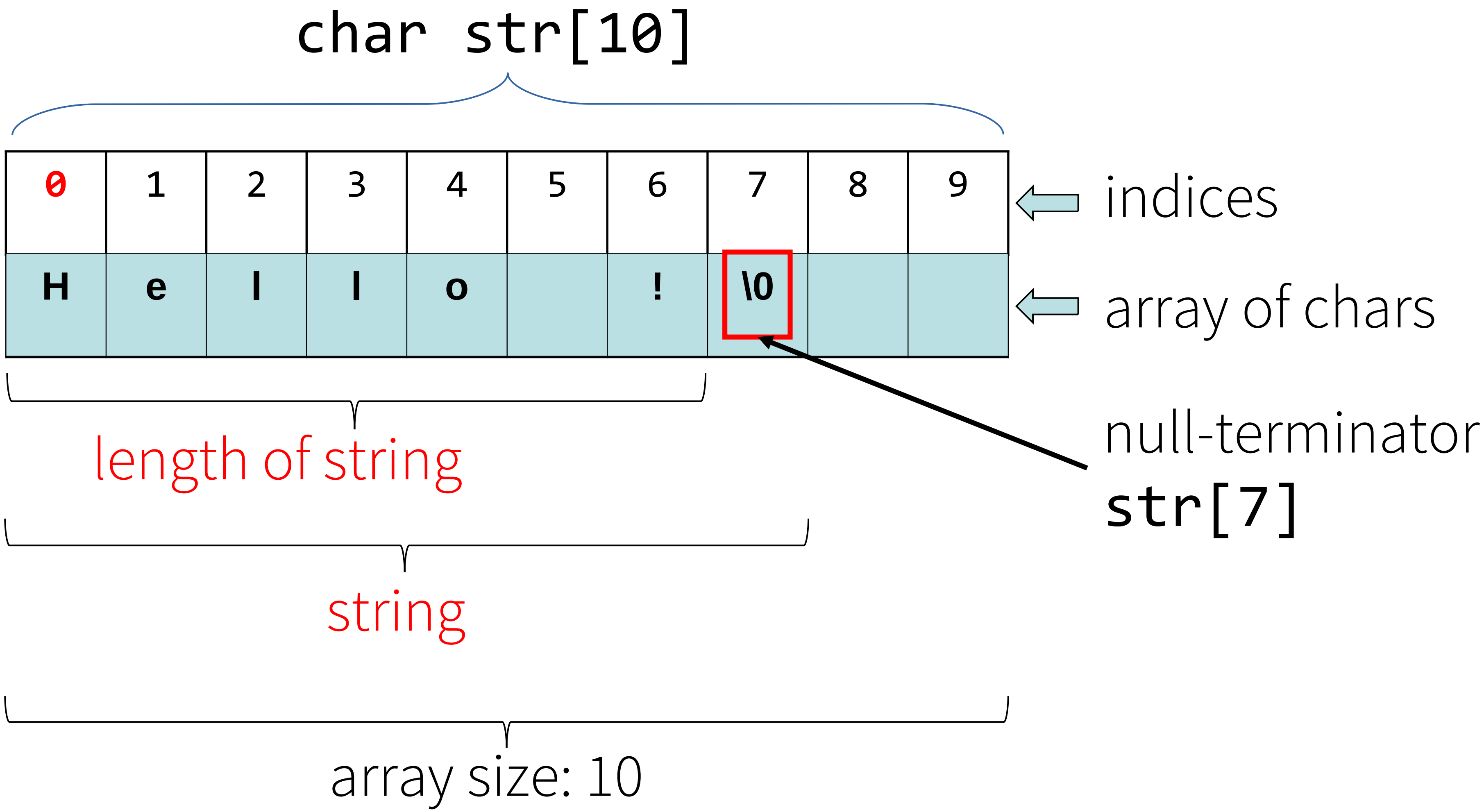
- We distinguish between two types of arrays:
  - One-dimensional arrays
  - Multi-dimensional arrays
    - The C/C++ language places no limits on the number of dimensions in an array, though specific implementations may

- The **size** and **type** of arrays **cannot** be changed after their declaration!

Arrays and C Strings

- A character array that contains ASCII characters terminated by the null character `'\0'` is a **C string variable**

```
char str[10] = "Hello !";
```



char str[10]

0	1	2	3	4	5	6	7	8	9
H	e	l	l	o		!	\0		

indices

array of chars

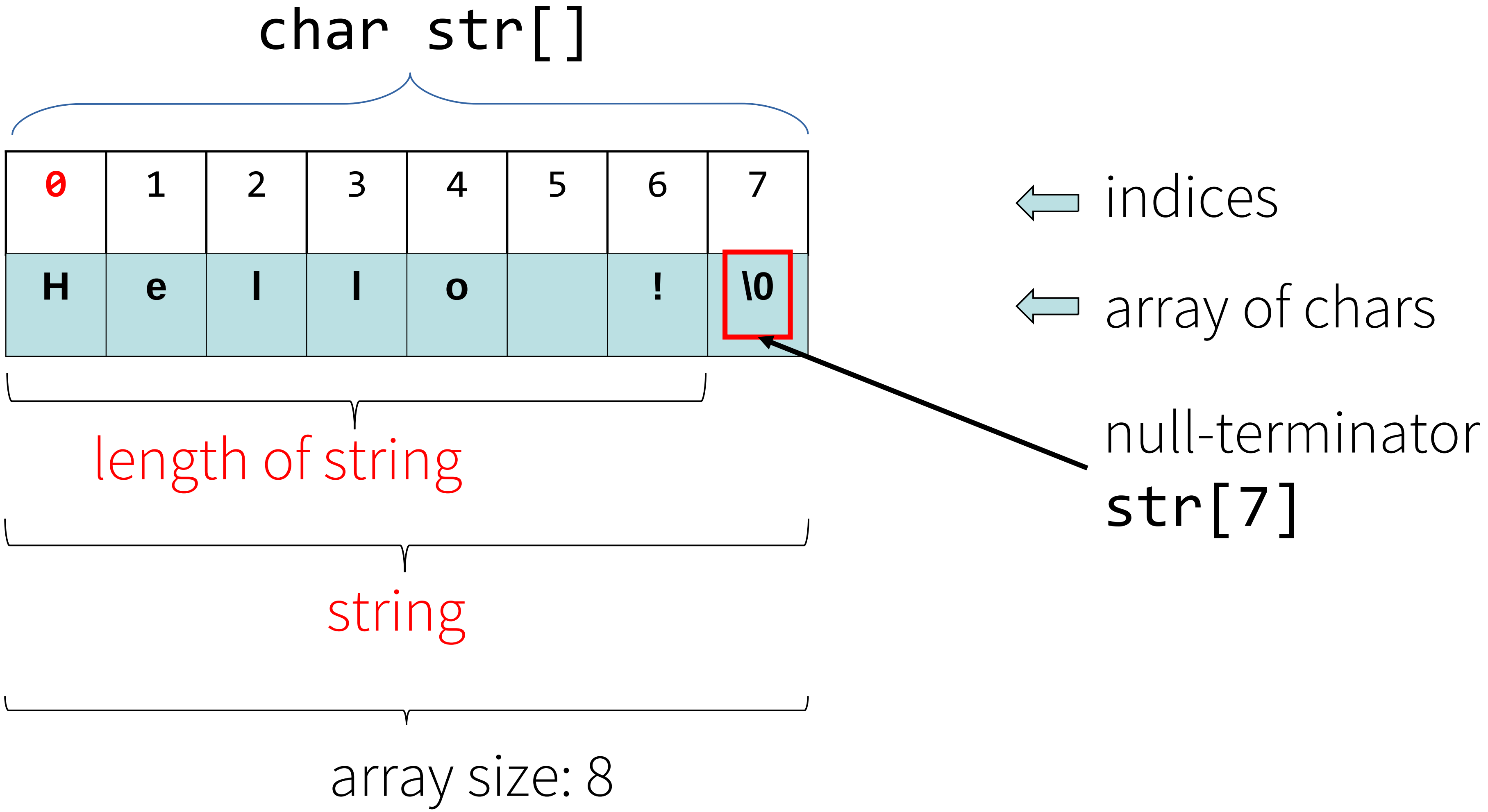
null-terminator
str[7]

length of string

string

array size: 10

```
char str[] = "Hello !";
```



char str[]

0	1	2	3	4	5	6	7
H	e	l	l	o		!	\0

indices

array of chars

null-terminator
str[7]

length of string

string

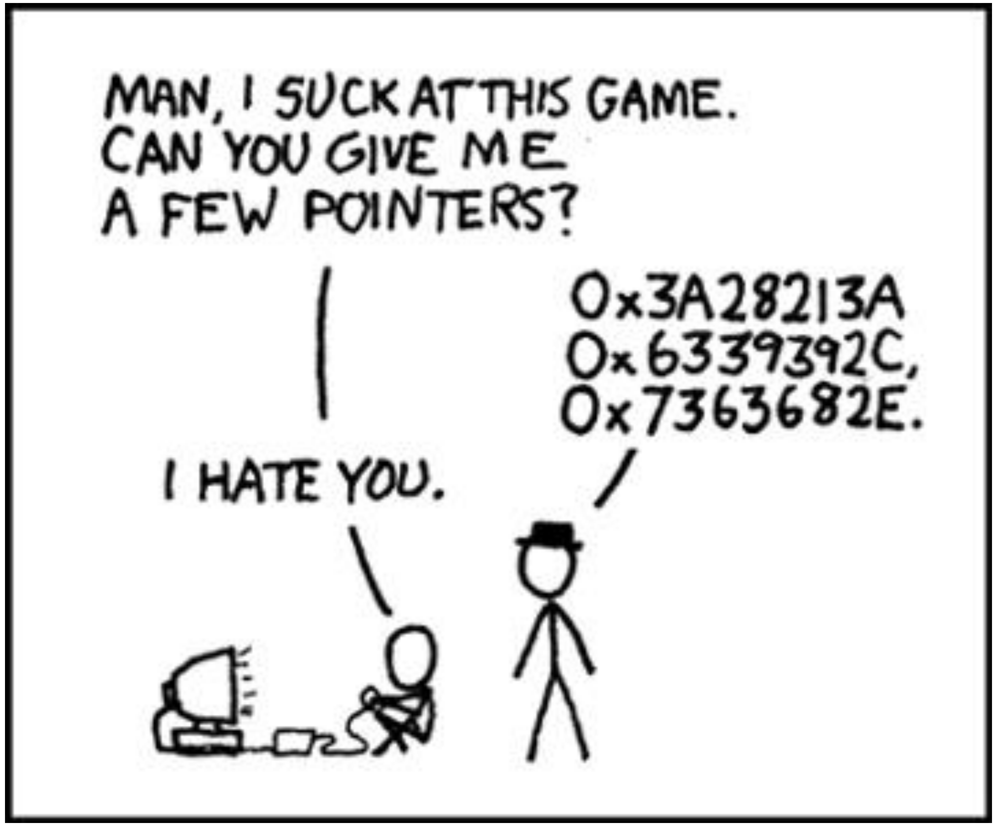
array size: 8

Pointers

Pointer: a variable that contain memory address as its value

Variable vs Pointer

- A variable directly refers to a value
- A pointer indirectly refers to a value



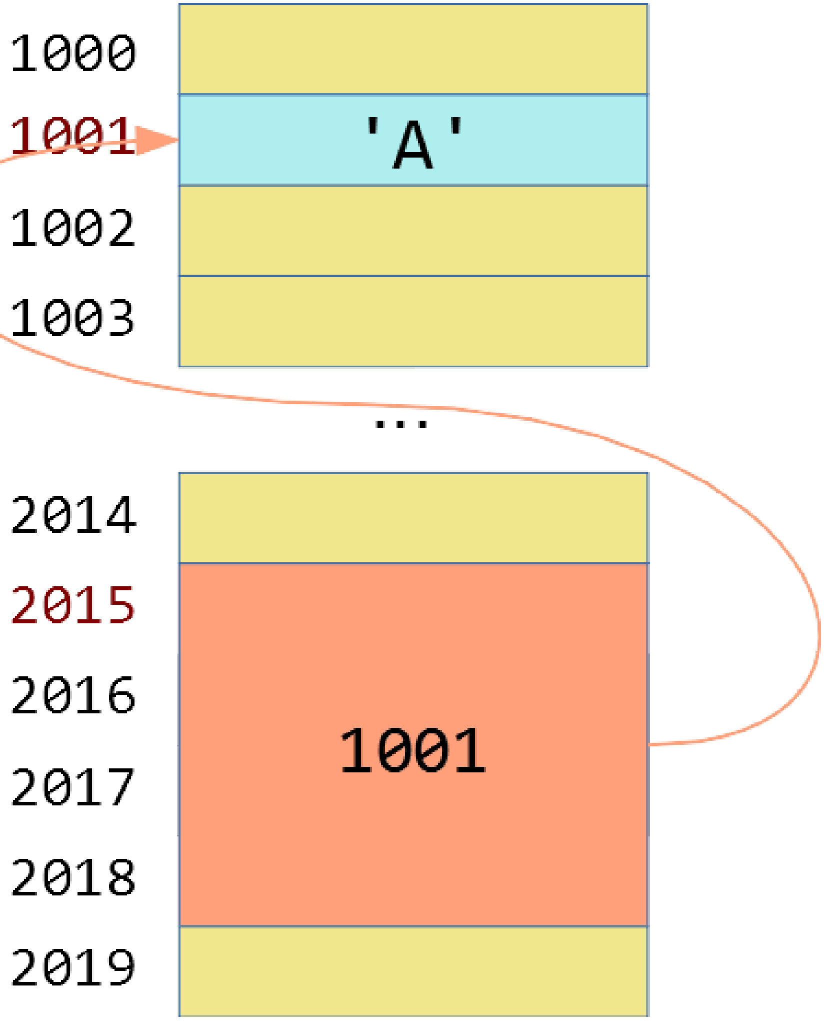
A pointer and a variable

```
char c = 'A';
```

A variable directly references a value

```
char *cp = &c;
```

A pointer indirectly references a value



Pointer Arithmetic

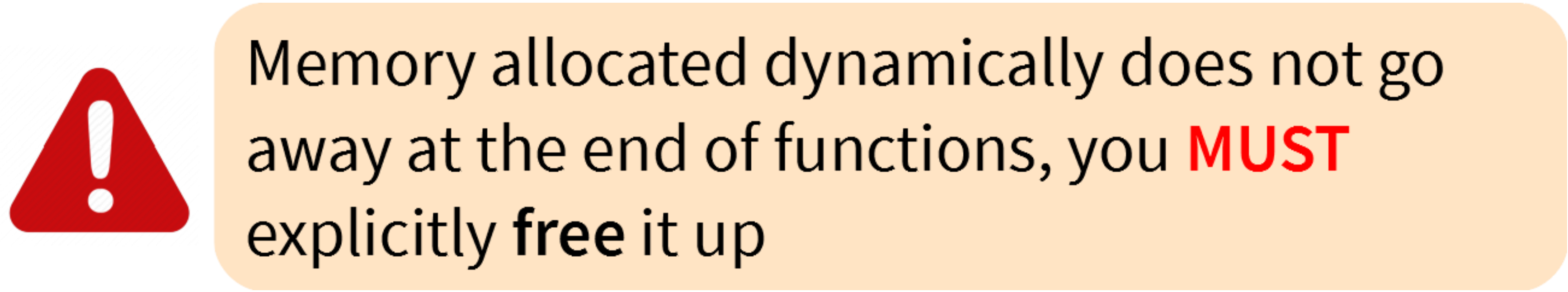
- Addition and subtraction can be performed on pointers, this is useful to iterate through arrays
- ++ Increments the pointer to the next element in the array
- -- decrements the pointer to the previous element in the array
- + adds the specified number of positions in the array to the pointer e.g. `pointer + 4` moves 4 elements in the array
- - subtracts the specified number of positions in the array from the pointer

Why Use Pointers?

- Provide an alternative means of accessing information stored in arrays, especially when working with strings
- To handle variable parameters passed to functions
- **To create dynamic data structures, that are built up from blocks of memory allocated from the heap at run time**

Dynamic Memory Allocation

- `calloc` - allocate arrays of memory
- `malloc` - allocate a single block of memory
- `realloc` - extend the amount of space allocated previously
- `free` - free up a piece of memory that is no longer needed by the program



Memory allocated dynamically does not go away at the end of functions, you **MUST** explicitly **free** it up

Dynamic Memory Allocation

- In addition to `calloc`, `malloc`, `realloc`, and `free`

- C++ has the new and `delete` keywords
  - new allocates memory on the heap
  - `delete` returns it to the OS
  - new and `delete` can be used with a wide range of data types

C Structure

- Syntax of the structure type declaration:

```
struct structure_tag {
     type1 member1;
     type2 member2;
     ...
} variable_list;
```

- Structure members can be
  - Basic data types
  - Derived and user-defined types
  - Pointers to basic, derived and user-defined data types

C++ Structure

- C++ structures are similar to C structures
- Same declaration syntax
- **But C++ structures can have functions as members and can be extended (supports inheritance)**
- **Member variables and functions are all public**

Classes

C++ classes generalizes structures in an object-oriented sense:

- Classes are types representing groups of similar instances
- Each instance has certain fields that define it (instance variables)
- Instances also have functions that can be applied to them– also known as *methods* in OOP parlance
- Access to parts of the class can be limited

Classes allow the combination of data and operations in a single unit

C++

Defining a Class

- A class is a collection of fixed number of components called **members** of the class

- General syntax for defining a class:

```
class class_identifier {
      class_member_list
};
```

  - `class_member_list` consists of variable declarations and/or functions

Example

```
class Time {
public:
        void set(int, int, int);
        void print() const;
        Time();
        Time(int, int, int);

private:
        int hour;
        int minute;
        int second;
};
```

Member access specifiers

Possible specifiers:

- `private`
- `protected`
- `public`

Example

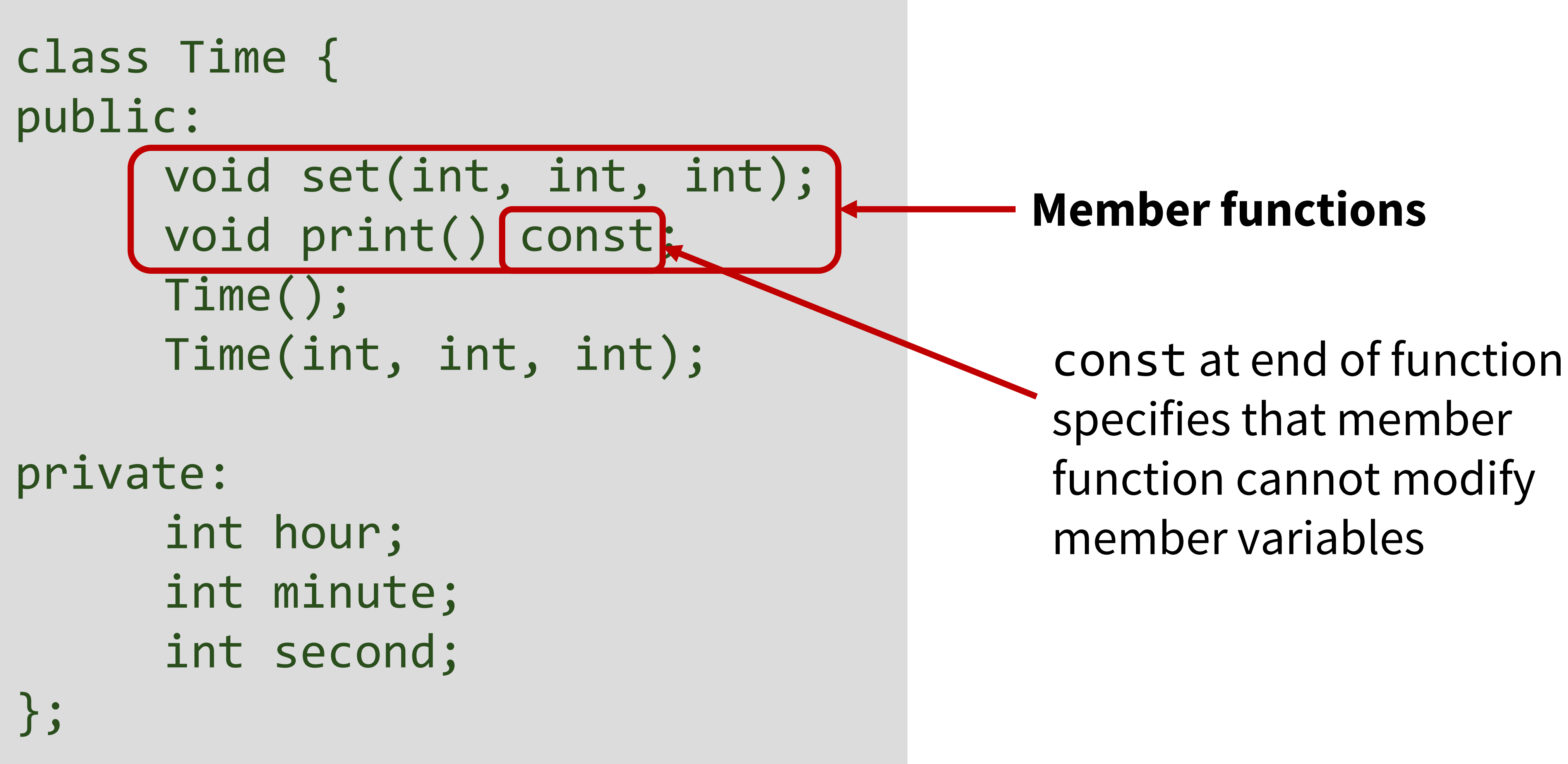
```
class Time {
public:
        void set(int, int, int);
        void print() const;
        Time();
        Time(int, int, int);

private:
        int hour;
        int minute;
        int second;
};
```

Constructors

- Named after class name
- Similar to Java

When class performs dynamic memory allocation, **destructor** is also needed

Example



```
class Time {
public:
      void set(int, int, int);
      void print() const;
      Time();
      Time(int, int, int);

private:
      int hour;
      int minute;
      int second;
};
```

Member functions

const at end of function specifies that member function cannot modify member variables

Example

```
class Time {
public:
      void set(int, int, int);
      void print() const;
      Time();
      Time(int, int, int);

private:
      int hour;
      int minute;
      int second;
};
```

Member variables

Constant member variables can be initialized during declaration

Extending Classes

- **Sub Class or Derived class** – a class that inherits member fields from another class
- **Super Class or Base Class** – a class whose fields are inherited by sub class
- **The sub class is said to extend the base class**

```
class subclass_name : access_mode baseclass_name {
     class_member_list
};
```

C++

Abstract Classes

- A class that contains at least one **pure virtual function** member
- Abstract classes cannot be instantiated
  - Similar to Java interfaces
- Pure virtual functions must be implemented by a sub class that need to be instantiated (**concrete)**

```
class Shape {
public:
  // Pure virtual function
  virtual float draw() = 0;

  // Virtual function
  virtual int getSides() {
    return 1;
  }
};
```

Final Exam

Time, Marks and Permitted Materials

- Time Allowed: 120 minutes
- Total Marks: 120
- Permitted Materials:
  - Only silent non-programmable calculators or silent programmable calculators with their memories cleared are permitted in this examination.
  - No electronic dictionaries are allowed.
  - Paper foreign to English language dictionaries are allowed.

Questions

- Short answer questions

- Topics:
    - C/C++ Fundamentals
    - User-Defined Types and C++ Classes
    - Arrays
    - Pointers
    - Dynamic Memory Allocation
    - C++ Templates and Vectors
    - Data Structures
    - File I/O
    - Low-Level and Socket Programming
    - Process Management

Good Luck!!!

That's all Folks!