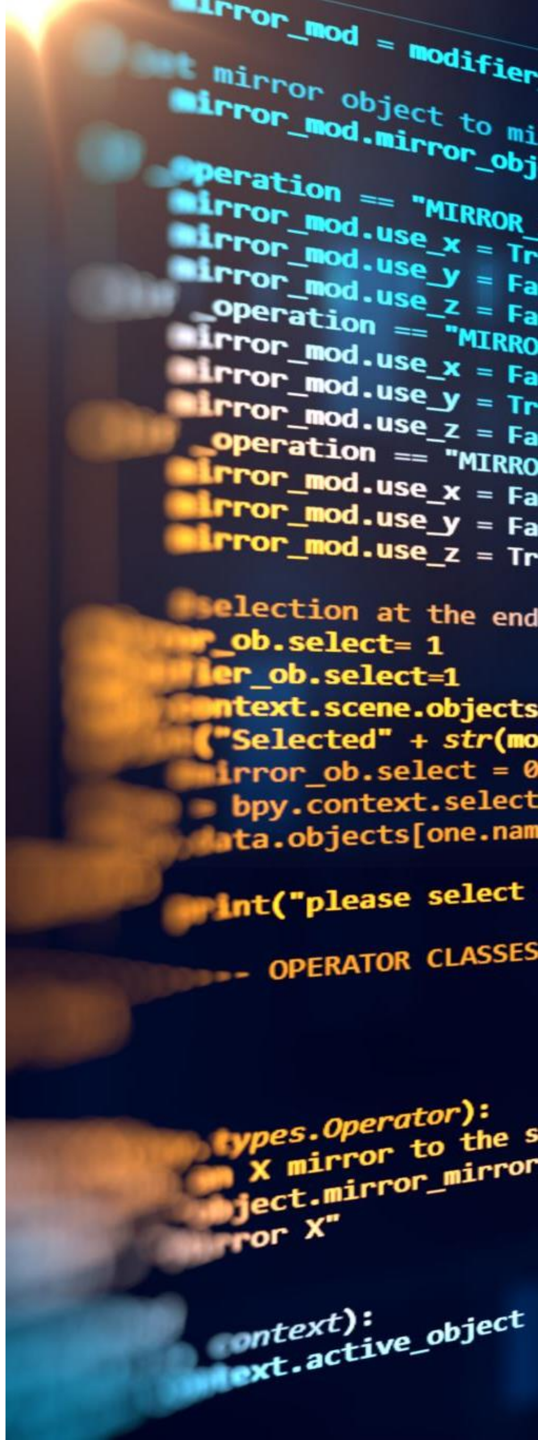
CS 1037
Fundamentals of Computer Science II

C Fundamentals (cont.)

Ahmed Ibrahim

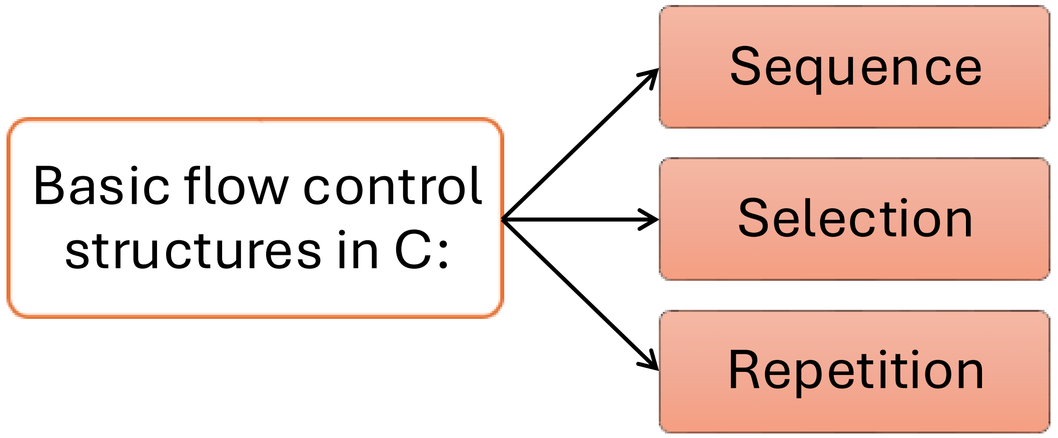


Recap

- **Data Types in C**
  - **Primary**: char, int, float
  - **User-defined**: enum, typedef
  - **Derived**: Pointers, arrays, and structures.
- **Variables**
  - A variable is a named memory location that stores values at runtime. C is a strongly typed language where every variable must be declared and initialized before use.
- **Scopes**
  - **Global scope**: Variables accessible throughout the program.
  - **Local scope**: Variables declared within a function or code block, accessible only within that scope.
- **Constants**
  - Constants are fixed values in a program, such as #define PI 3.14159.
  - These can also be defined using the const keyword to declare a variable as read-only.
- **Operations and Expressions**
  - Arithmetic operations: Addition (+), subtraction (-), multiplication (*), division (/), and modulus (%).
  - Relational (==, !=, <, >, etc.) and logical operators (&&, ||, !) allow for more complex expressions and conditional logic.

Flow Controls in C

- Flow controls determine the order of execution of statements in a program.
- Three basic flow control structures in C:



Selection Control

- **Selection control** alters the default flow based on conditions.
- Types of decision control statements in C:
  - if statement
  - if-else statement
  - if-else-if statement
  - switch statement

Decision Control Example

- This program uses **IF**, **IF-ELSE**, and **IF-ELSE-IF** statements to evaluate a number's size, check even/odd, and assign a grade based on the input value.

```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter a number (1-100): ");
    scanf("%d", &num);

    // if statement
    if (num > 50) printf("Greater than 50\n");

    // if-else statement
    (num % 2 == 0) ? printf("Even\n") : printf("Odd\n");

    // if-else-if statement
    if (num >= 90) printf("Grade: A\n");
    else if (num >= 80) printf("Grade: B\n");
    else if (num >= 70) printf("Grade: C\n");
    else if (num >= 60) printf("Grade: D\n");
    else printf("Grade: F\n");

    return 0;
}
```

User Input

Condition

User Input in C

- The scanf() function is used to take formatted input from the user.
- It reads data from the standard input (**stdin**) and stores it in the corresponding variable.
- Syntax: `scanf("format_specifier", &variable);`
    - **format_specifier**: Defines the type of input to be read.
    - **&variable**: Address of the variable where the input will be stored.
- Common Format Specifiers:
    - **%d** – Reads an integer
    - **%f** – Reads a float
    - **%c** – Reads a single character
    - **%s** – Reads a string (without spaces)

SWITCH Example

- Using specific cases, this program uses a **switch** statement to display feedback based on the input grade (A, B, C, D, or F).
- The **break** statement prevents a fall-through to the next case.
- Without a **break**, execution would continue to the next case.

```c
#include <stdio.h>

int main() {
    char grade;
    printf("Enter a grade (A, B, C, D, F): ");
    scanf(" %c", &grade);

    switch (grade) {
        case 'A':
            printf("Excellent!\n");
            break;
        case 'B':
            printf("Very Good!\n");
            break;
        case 'C':
            printf("Good\n");
            break;
        case 'D':
            printf("Pass\n");
            break;
        case 'F':
            printf("Fail\n");
            break;
        default:
            printf("Invalid grade\n");
    }

    return 0;
}
```

SWITCH Syntax

```
switch(expression) {
    case value1:
        // Code to execute if expression == value1
        break;
    case value2:
        // Code to execute if expression == value2
        break;
    ...
    default:
        // Code to execute if no matching case
}
```

Repetition Control and Loop Types in C

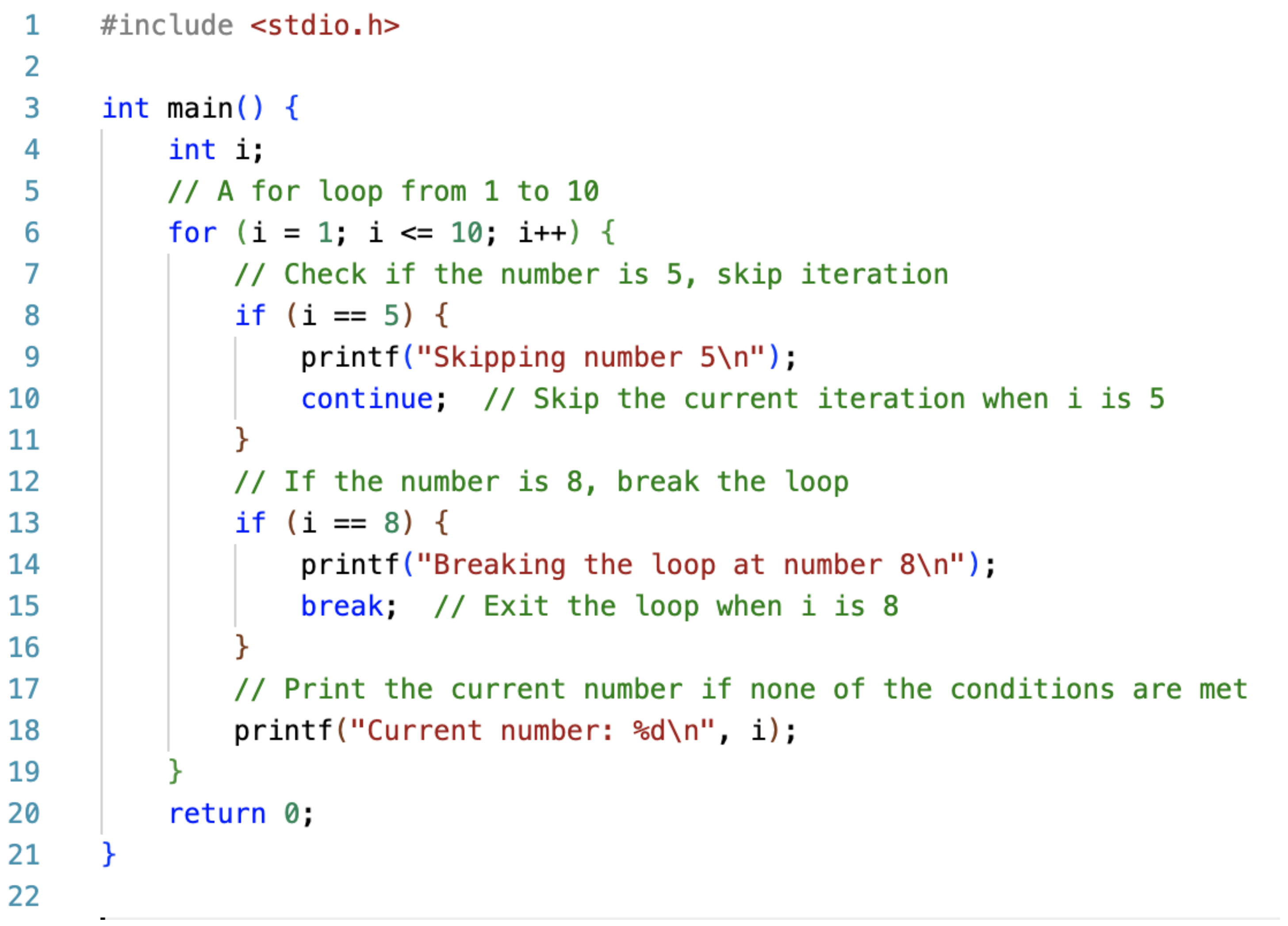
- Repetition Control repeats a block of statements until a condition is satisfied.
- Types of loops in C:
  - FOR loop
  - WHILE loop
  - DO-WHILE loop

- Difference Between while and do-while Loops:
  - WHILE loop: Checks the condition before executing the block.
  - DO-WHILE loop: Executes the block first, then checks the condition.

- Other Control Statements in C:
  - BREAK statement: Exits the current loop or switch statement.
  - CONTINUE statement: Skips the current iteration and continues with the next.
  - GOTO statement: A flexible control statement allowing jumps to labeled parts of the code.

Repetition Control

- This C program uses a for loop, along with if-else, continue, and break statements, to skip number 5 and terminate the loop at number 8.



```c
#include <stdio.h>

int main() {
    int i;
    // A for loop from 1 to 10
    for (i = 1; i <= 10; i++) {
        // Check if the number is 5, skip iteration
        if (i == 5) {
            printf("Skipping number 5\n");
            continue;  // Skip the current iteration when i is 5
        }
        // If the number is 8, break the loop
        if (i == 8) {
            printf("Breaking the loop at number 8\n");
            break;  // Exit the loop when i is 8
        }
        // Print the current number if none of the conditions are met
        printf("Current number: %d\n", i);
    }
    return 0;
}

```

Question?

Which of the following is true about the do-while loop in C?

A) The loop condition is checked before the loop body is executed.

B) The loop body is guaranteed to execute at least once, regardless of the condition.

C) It is equivalent to the while loop in functionality.

D) It always runs indefinitely.

Functions

- Function is the fundamental feature of C programming language. Basically, a C program consists of a collection of functions related by calling dependences.
- Using functions has the following advantages:
  - **Code reuse.** It is better to use functions for frequently used blocks of code. Write once and use it many times.
  - **Support modular and structured program design.** When dealing with a large and complex program, it is practical to decompose it into many smaller parts for effective development and maintenance.

```c
/*
C program structure example
*/
#include<stdio.h>         // preprocessor directive include
int a;                    // global variable declaration
int add(int, int);        // function declaration
int minus(int, int);      // function declaration
int main()                // main function
{
  a=1;                    // assign/set value 1 to global variable a
  int b=2;                // declare local variable b and initialize/set it to value 2
  printf("a+b=%d\n", add(a, b));   // function calls
  printf("a-b=%d\n", minus(a, b)); // function calls
  return 0;
}
// definition/implementation of function add(int, int)
int add(int x, int y)     // function header
{
   return x+y;            // function body
}
// definition/implementation of function minus(int, int)
int minus(int x, int y)   // function header
{
  return x-y;             // function body
}
```

Function Inputs and Outputs

- Two methods to pass inputs:
  - Pass-by-value: Passes a copy of the data.
  - Pass-by-reference: Passes the memory address of the variable.
- Output can be returned by:
  - Function return value.
  - Pass-by-reference.
  - Global variables (less commonly used).

Pass-by-value

- Copies values of variables into function parameters.
- Operations inside the function do not affect variables outside the function.

```
int max(int x, int y)
   {
      return x > y ? x : y;
   }
```

Ternary operator (also known as the **conditional operator**) in C.

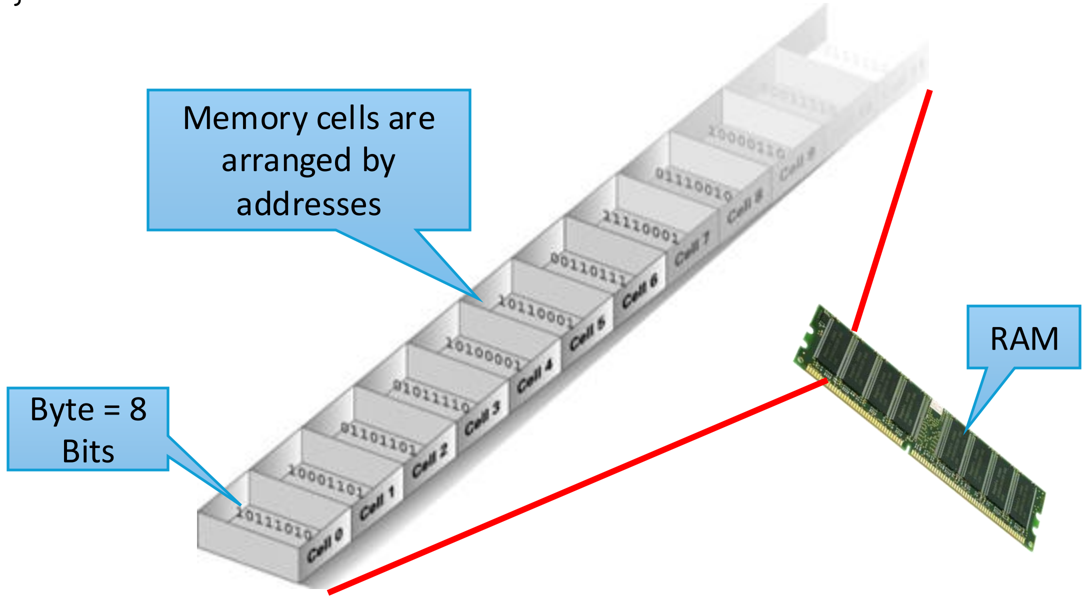
```
if (x > y) {
   return x;
} else {
   return y;
}
```

Pass-by-Reference

- Passes addresses (references) of variables to the function.
- Allows the function to modify external variables.

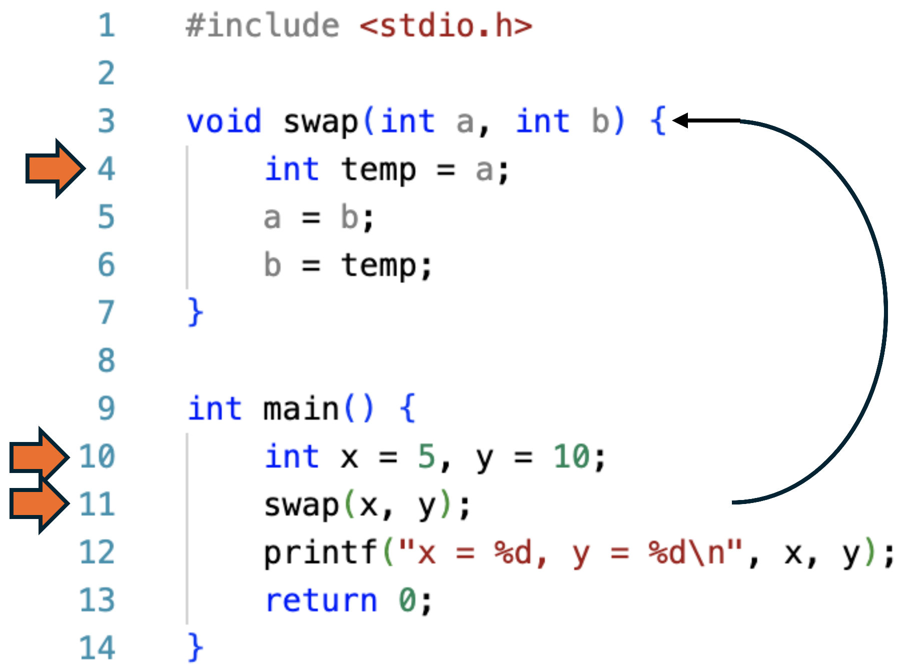
Note: A computer's memory consists of a set of memory cells, each having a **unique address**.

```
void inc(int *x)
{
     *x = *x + 1;
  }
```



Question?

- Consider the following code:



```c
#include <stdio.h>

void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 5, y = 10;
    swap(x, y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

What will be the x and y output after the function call?

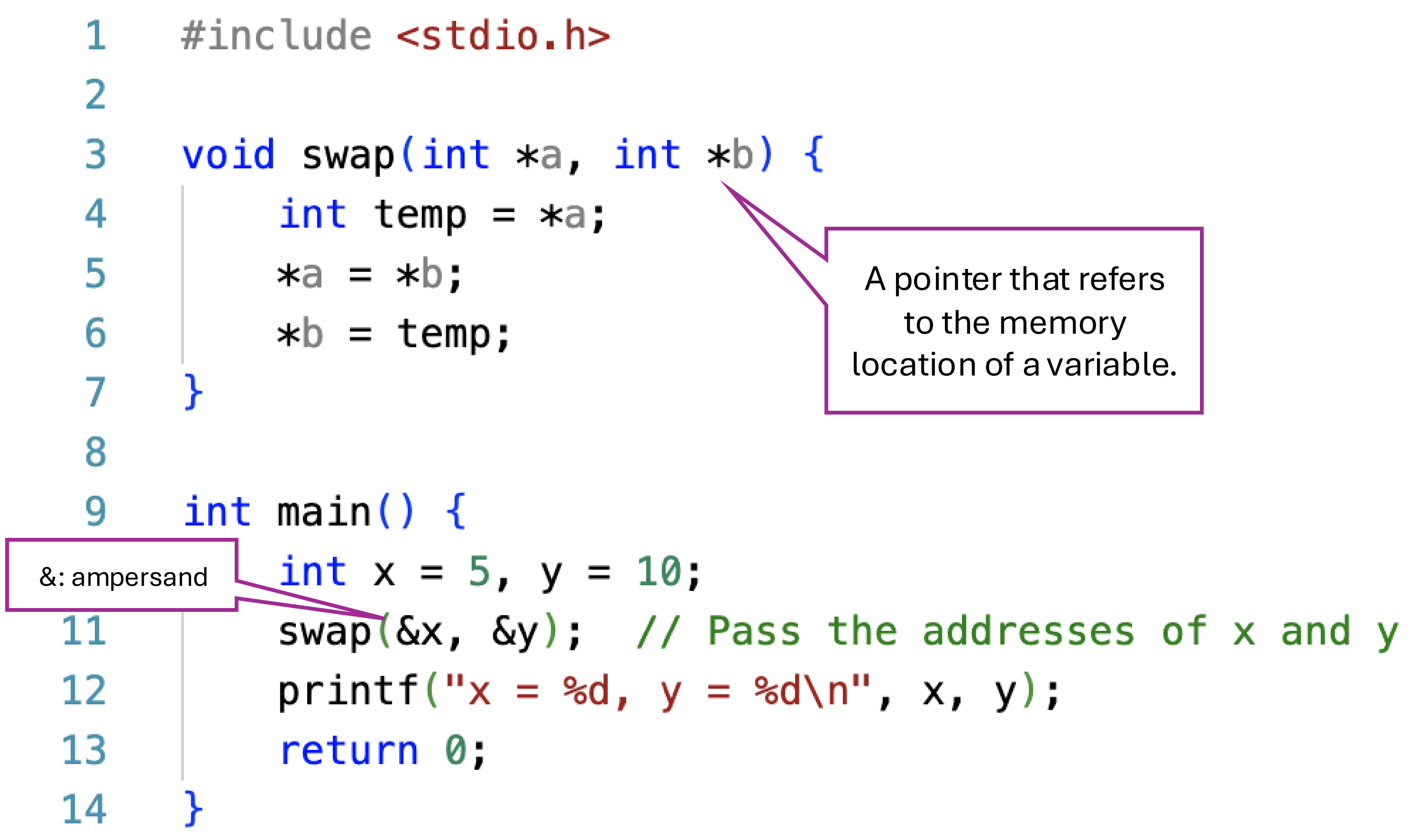
A) x = 10, y = 5

B) x = 5, y = 10

C) x = 0, y = 0

D) Swap will fail because pass-by-value cannot modify the variables.

Pass-by-Reference Example



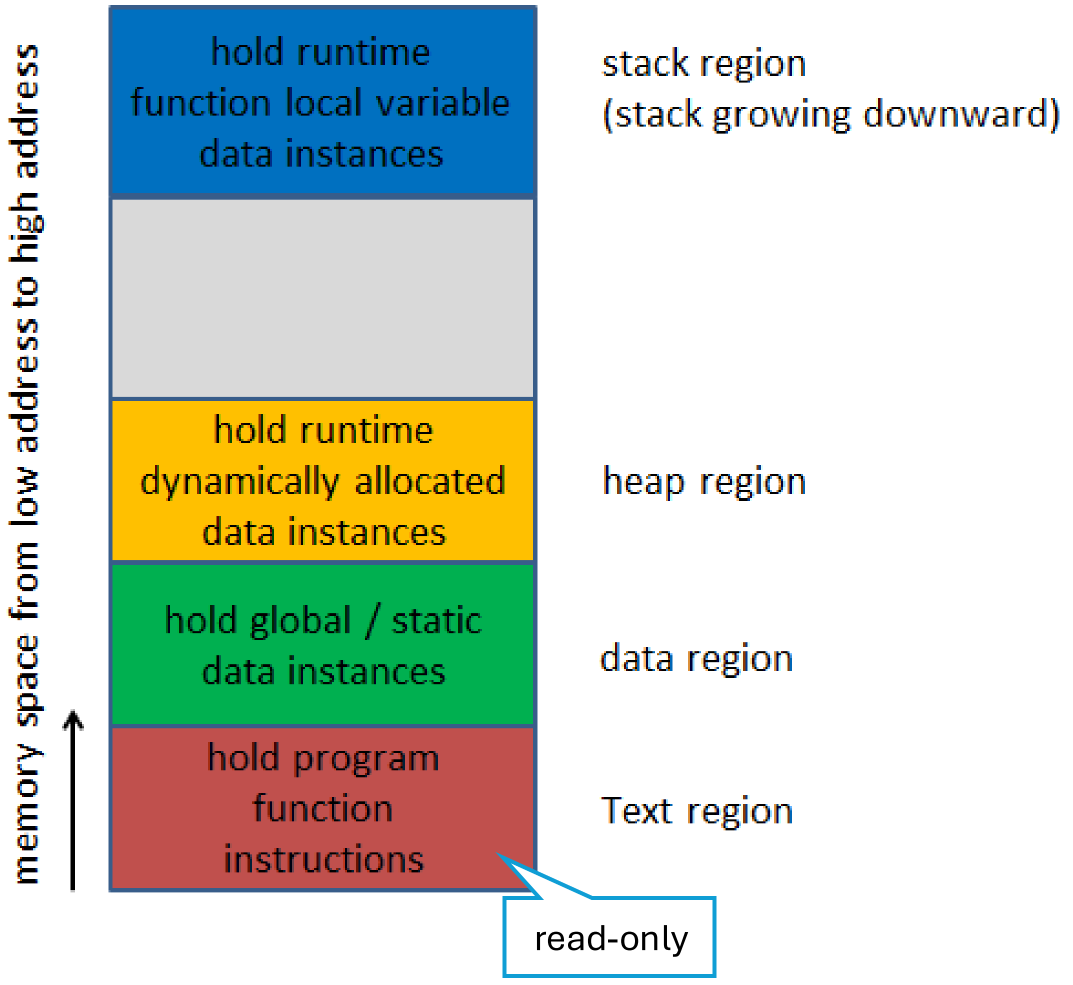
```c
#include <stdio.h>

void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main() {
    int x = 5, y = 10;
    swap(&x, &y);  // Pass the addresses of x and y
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

A pointer that refers to the memory location of a variable.

&: ampersand

Memory management of program executions

- An executable program consists of a sequence of instructions organized by functions.
- Each instruction consists of a fixed number of bytes (4 bytes in a 32-bit system, 8 bytes in a 64-bit system).
- Memory is assigned by the OS for running a program.
- The memory space consists of:
  - **Program memory**: Stores function instructions in the text region (**text segment or region**).
  - **Data memory**:
    - **Data region**: Stores static and global variables.
    - **Stack region**: Stores parameters and local variables when a function is called.
    - **Heap region**: Stores dynamically allocated memory blocks.



memory space from low address to high address
hold runtime function local variable data instances
stack region (stack growing downward)
hold runtime dynamically allocated data instances
heap region
hold global / static data instances
data region
hold program function instructions
Text region
read-only

Thank you

References

- Data Structures Using C, second edition, by Reema Thareja, Oxford University Press, 2014.