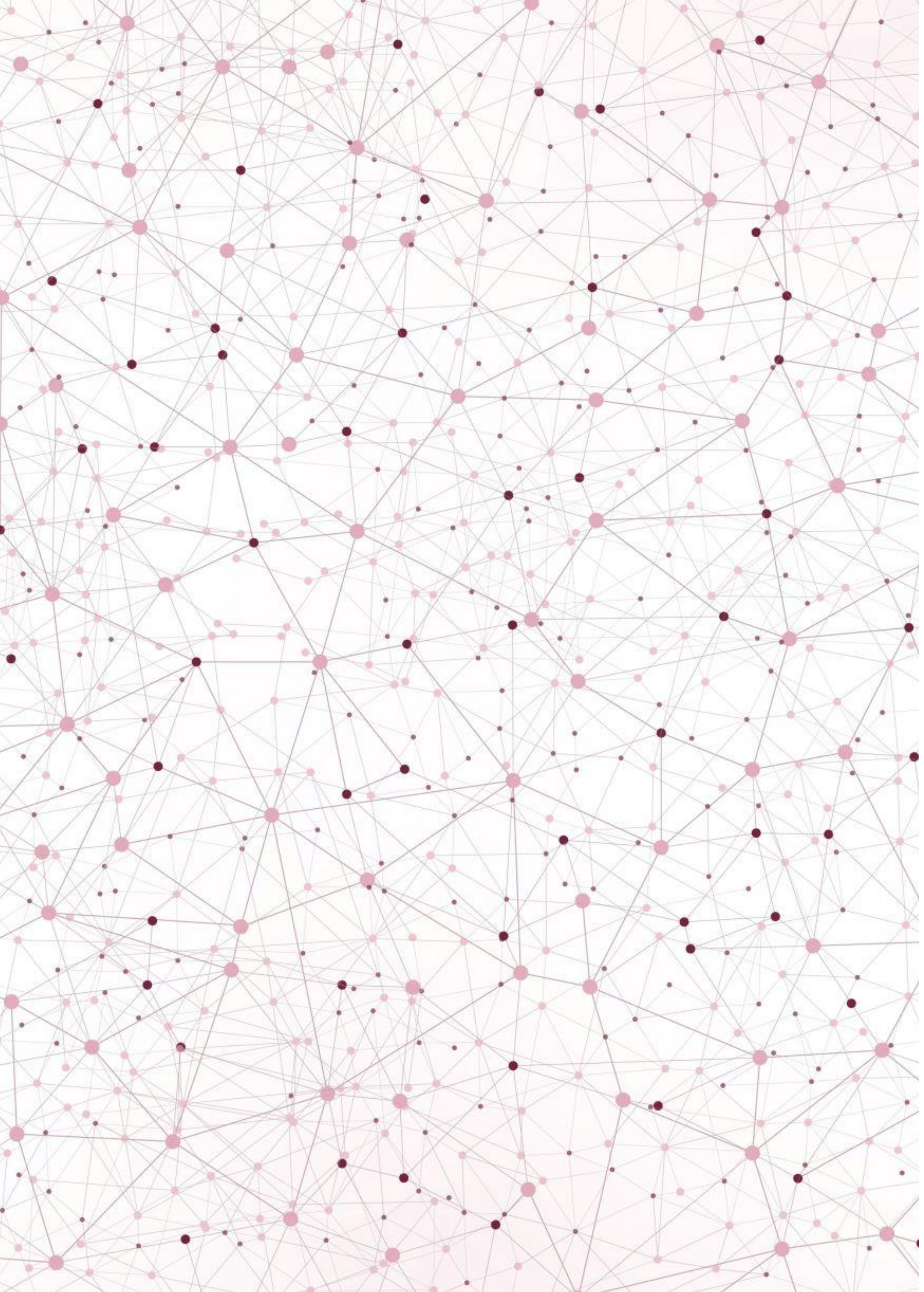


PROGRAMMING LANGUAGES C

Medvei Mirabela

2025

SETUP LINUX VM

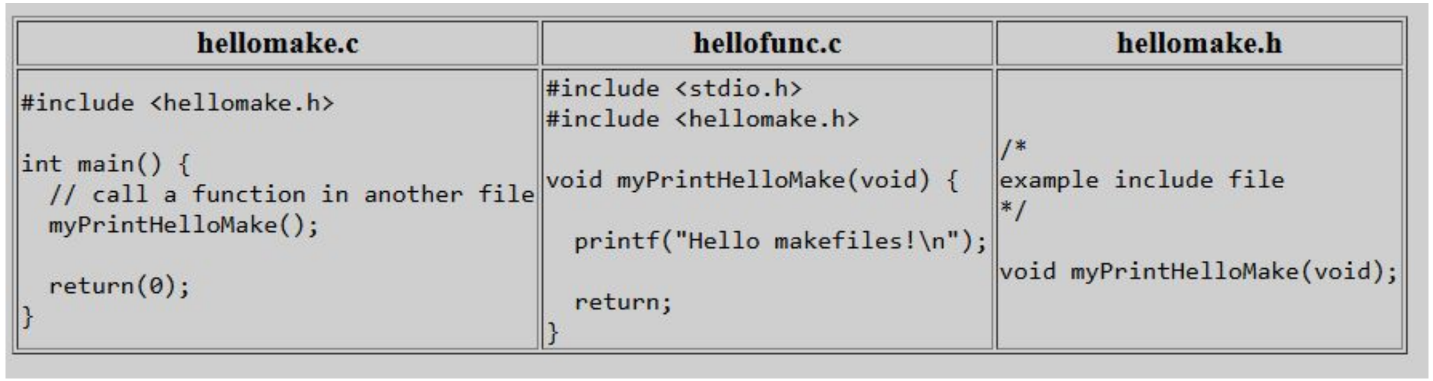
- Download and install the Ubuntu image https://ubuntu.com/download/desktop/thank-you?version=24.04.2&architecture=amd64&lts=true
- Install latest gcc:
  - *$ sudo apt-get update*
  - *$ sudo apt-get install gcc make*
  - *$ gcc –v to see the gcc installed version*
  - *$ make -v*
- Install Visual Studio Code. Example installation of Visual Studio Code for Ubuntu: https://code.visualstudio.com/docs/setup/linux
  - Download from https://code.visualstudio.com/download. For Ubuntu, select the ".deb" file. Then run the following:
    - *$ sudo apt-get update*
    - *$ sudo dpkg -i filename.deb*
  - You can search and open the *"Visual Studio Code"* application from the graphical interface or run the "*code*" command in the terminal.

MAKEFILE

- *Make* tool - automatically determines which parts of a project need to be recompiled as a result of modifications and triggers the necessary commands for recompiling them. In order to use make, a **makefile** (usually called Makefile or makefile) is required.
- This file describes the dependency relationships between the various files that make up the program and specifies the update rules for each individual file.
- Example of makefile: https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

MAKEFILE - EXAMPLE

- The following files, **hellomake.c**, **hellofunc.c**, and **hellomake.h**, represent a typical main program, some functional code in a separate file, and an include file, respectively.



hellomake.c	hellofunc.c	hellomake.h

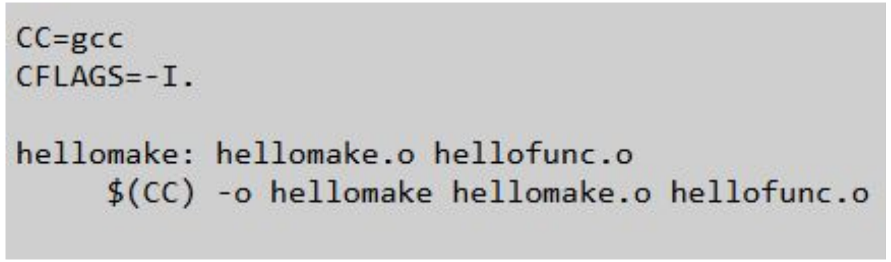
`#include <hellomake.h>`
 	
`int main() {`
`  // call a function in another file`
`  myPrintHelloMake();`
 	
`  return(0);`
`}`
	`#include <stdio.h>`
	`#include <hellomake.h>`
	
	`void myPrintHelloMake(void) {`
	
	`  printf("Hello makefiles!\n");`
	
	`  return;`
	`}`
		`/*`
		`example include file`
		`*/`
		
		`void myPrintHelloMake(void);`

- Normally, you would compile this collection of code by executing the following command:
  - $ *gcc -o hellomake hellomake.c hellofunc.c -I.*

  (This compiles the two .c files and names the executable hellomake. The -I. is included so that gcc will look in the current directory (.) for the include file hellomake.h.)

MAKEFILE - EXAMPLE

- The simplest makefile you could create would look something like:



```
CC=gcc
CFLAGS=-I.

hellomake: hellomake.o hellofunc.o
	$(CC) -o hellomake hellomake.o hellofunc.o
```

- If you put this rule into a file called **Makefile** or **makefile** and then type make on the command line it will execute the compile command as you have written it in the makefile. Note that make with no arguments executes the first rule in the file.

- There must be **a tab** at the beginning of any command, and make will not be happy if it's not there.

OBJECTIVES - ASSESSING YOUR C PROGRAMMING SKILLS

These labs will give you practice in the style of programming you will need to be able to do proficiently, especially for the later assignments in the class. The material covered should all be review for you. Some of the skills tested are:

- Explicit memory management, as required in C.
- Creating and manipulating pointer-based data structures.
- Working with strings.
- Enhancing the performance of key operations by storing redundant information in data structures.
- Implementing robust code that operates correctly with invalid arguments, including NULL pointers.

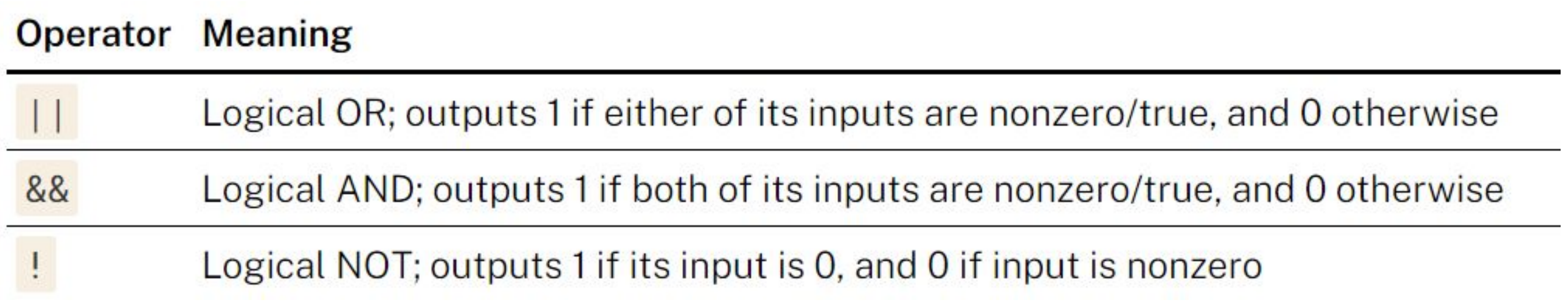
RECAP: BITWISE AND LOGICAL OPERATORS

Operator	Meaning
&	Bitwise AND
|	Bitwise inclusive OR
^	Bitwise exclusive OR
<<	Left shift
>>	Right shift
~	One's complement / bitwise NOT (unary)

You can also freely write numbers in hexadecimal (recall **0x** means the following number is hexadecimal), and print numbers in hexadecimal using the **%x format in printf.**

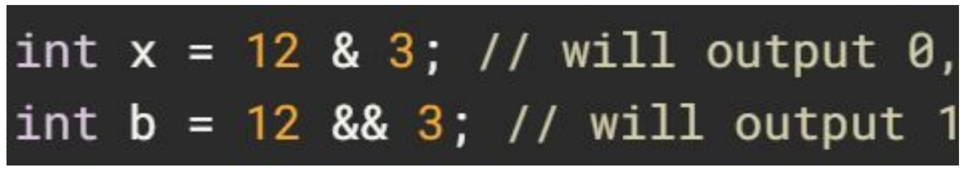
```
int x = 0x4f3e;
printf("%x", x);
```

RECAP: BITWISE AND LOGICAL OPERATORS



Operator	Meaning
`||`	Logical OR; outputs 1 if either of its inputs are nonzero/true, and 0 otherwise
`&&`	Logical AND; outputs 1 if both of its inputs are nonzero/true, and 0 otherwise
`!`	Logical NOT; outputs 1 if its input is 0, and 0 if input is nonzero

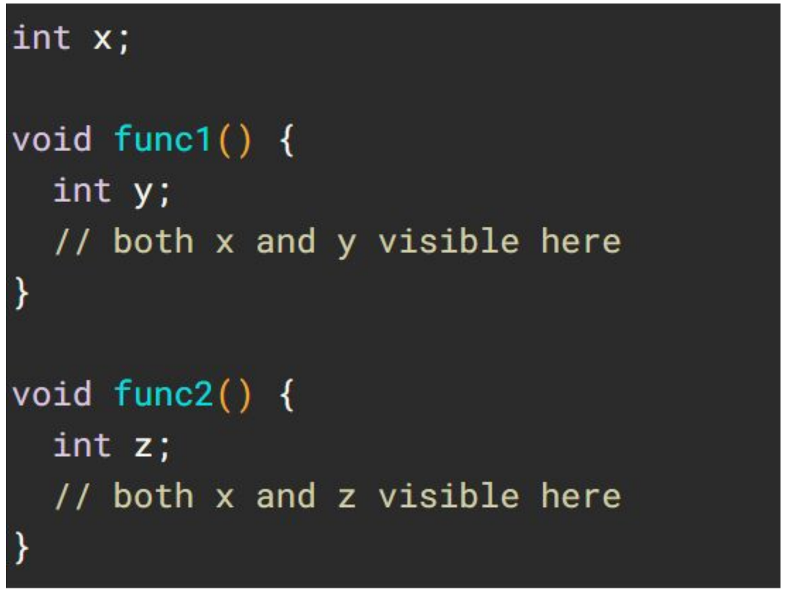
The C spec deals with this by treating **ALL nonzero numbers as true**. So something like 2 && 3 would be interpreted as TRUE && TRUE, and so output 1 which means TRUE.



```
int x = 12 & 3; // will output 0,
int b = 12 && 3; // will output 1
```

RECAP: SCOPES OF VARIABLES

- Let it be noted that variables defined inside functions are only **visible inside the function(local variables)**. You can also have variables that sit in the "top level" program outside all of the functions, which **are global and visible in all functions.**



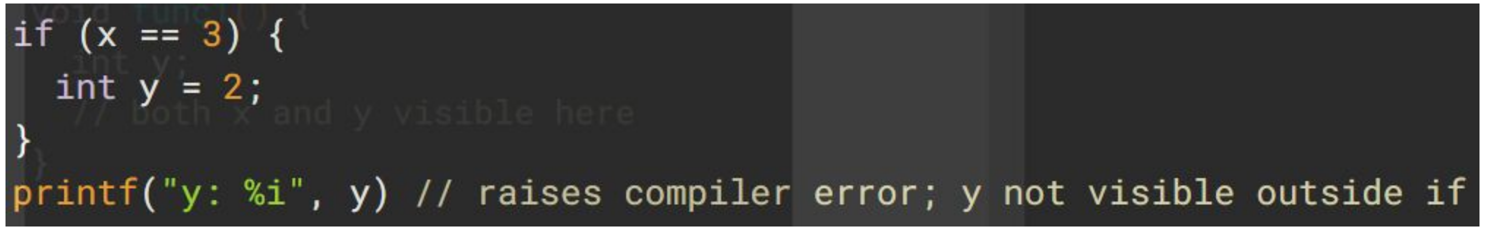
```
int x;

void func1() {
  int y;
  // both x and y visible here
}

void func2() {
  int z;
  // both x and z visible here
}
```

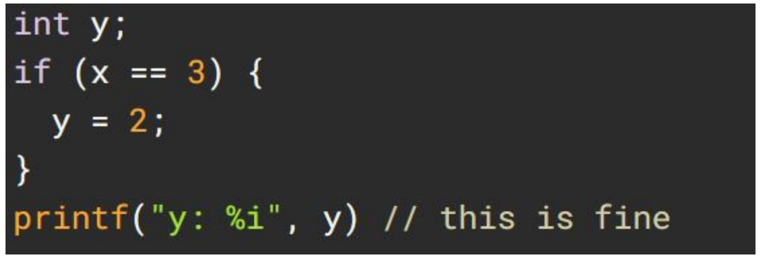
RECAP: SCOPES OF VARIABLES

- Control flow statements also define inner scopes in their clauses. For example, a variable defined in an if **statement is not visible outside of it:**



```c
if (x == 3) {
  int y = 2;
}
printf("y: %i", y) // raises compiler error; y not visible outside if
```

- Instead, we need to define the variable outside the if statement and only ***initialise*** it in the if statement:



```c
int y;
if (x == 3) {
  y = 2;
}
printf("y: %i", y) // this is fine
```

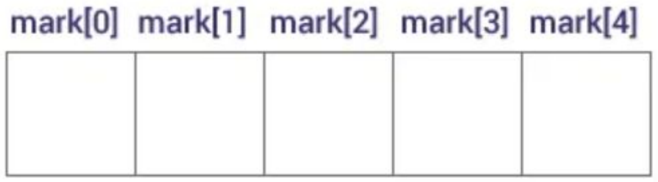
RECAP: ARRAYS

- An array is a variable that can store multiple values. For example, if you want to store 100 integers, you can create an array for it.

```
int data[100];
```

- You can access elements of an array by indices.

```
float mark[5];
```



RECAP: ARRAYS

- How to initialize an array?
  - It is possible to initialize an array during declaration. For example:

```
int mark[5] = {19, 10, 8, 17, 9};
```

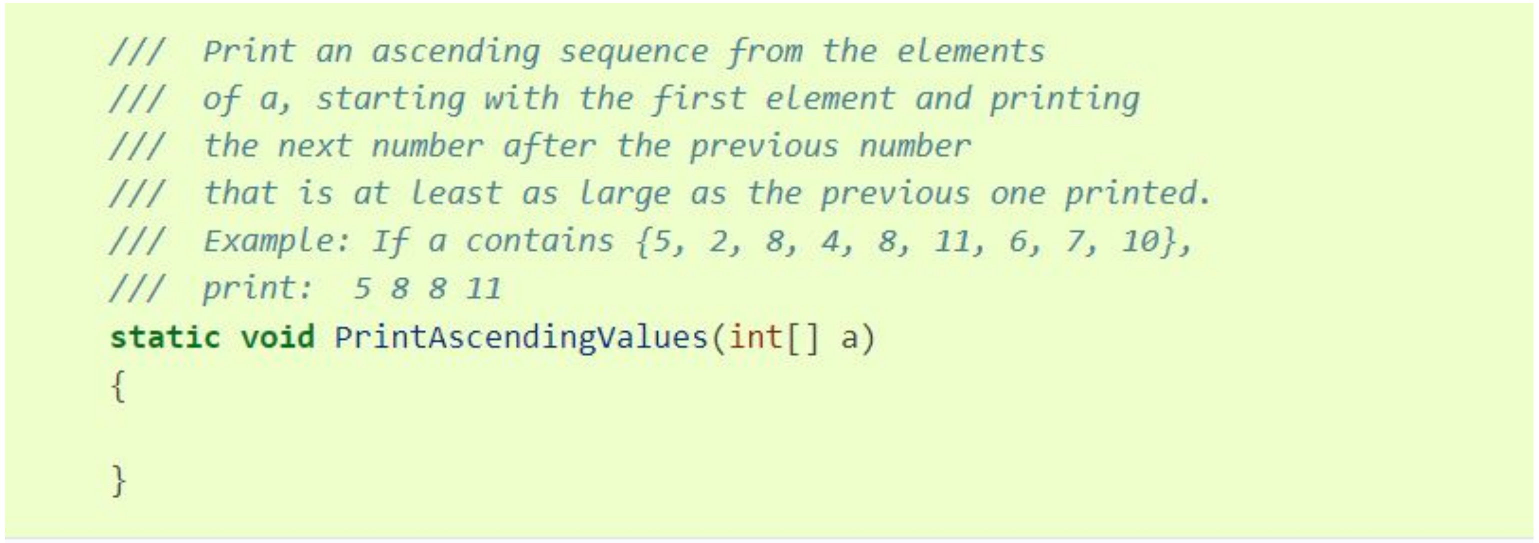
  - You can also initialize an array like this:

```
int mark[] = {19, 10, 8, 17, 9};
```

  - Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

RECAP: ARRAYS - EXERCISES

1. Write a program in C to read n number of values in an array and display them in reverse order.
2. Write a program in C to merge two arrays of the same size sorted in descending order.
3. Complete and test the function with documentation and heading:

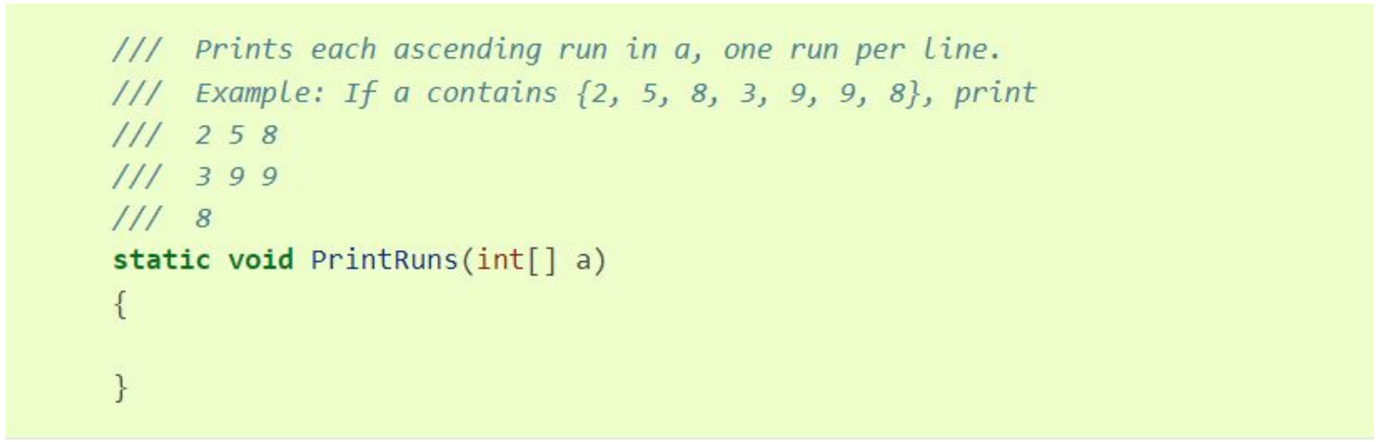


```
/// Print an ascending sequence from the elements
/// of a, starting with the first element and printing
/// the next number after the previous number
/// that is at least as large as the previous one printed.
/// Example: If a contains {5, 2, 8, 4, 8, 11, 6, 7, 10},
/// print:  5 8 8 11
static void PrintAscendingValues(int[] a)
{

}
```

RECAP: ARRAYS - EXERCISES

4. Complete and test the function with documentation and heading:

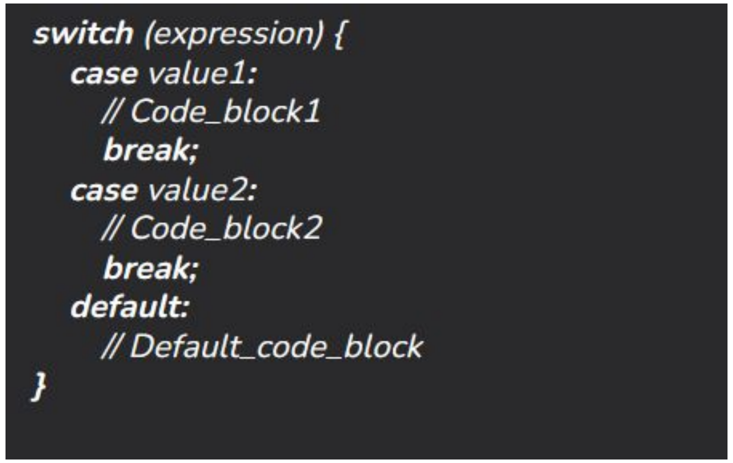


```
///  Prints each ascending run in a, one run per line.
///  Example: If a contains {2, 5, 8, 3, 9, 9, 8}, print
///  2 5 8
///  3 9 9
///  8
static void PrintRuns(int[] a)
{

}
```

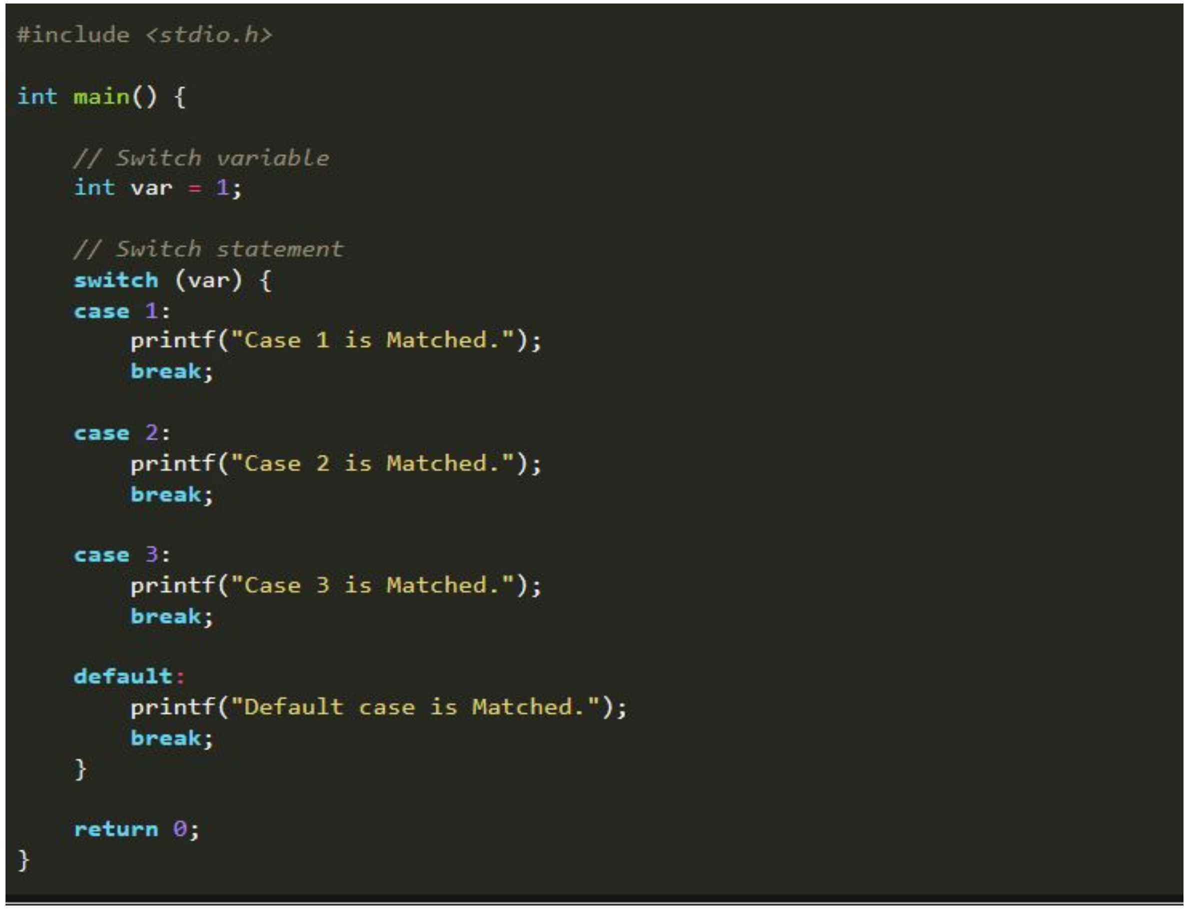
SWITCH STATEMENT

- In C, switch statement is a control flow structure that allows you to execute one of many code blocks based on the value of an expression. It is often used in place of if-else ladder when there are multiple conditional codes.
- Syntax of switch:



```
switch (expression) {
   case value1:
      // Code_block1
      break;
   case value2:
      // Code_block2
      break;
   default:
      // Default_code_block
}
```

SWITCH STATEMENT - EXAMPLE



```c
#include <stdio.h>

int main() {

    // Switch variable
    int var = 1;

    // Switch statement
    switch (var) {
    case 1:
        printf("Case 1 is Matched.");
        break;

    case 2:
        printf("Case 2 is Matched.");
        break;

    case 3:
        printf("Case 3 is Matched.");
        break;

    default:
        printf("Default case is Matched.");
        break;
    }

    return 0;
}
```