LBPS : C Basics

Preparatory Module

Course Outline

1. Introduction to C

2. Control statements

3. Functions

4. Arrays

5. Strings

Lesson 1 : topics

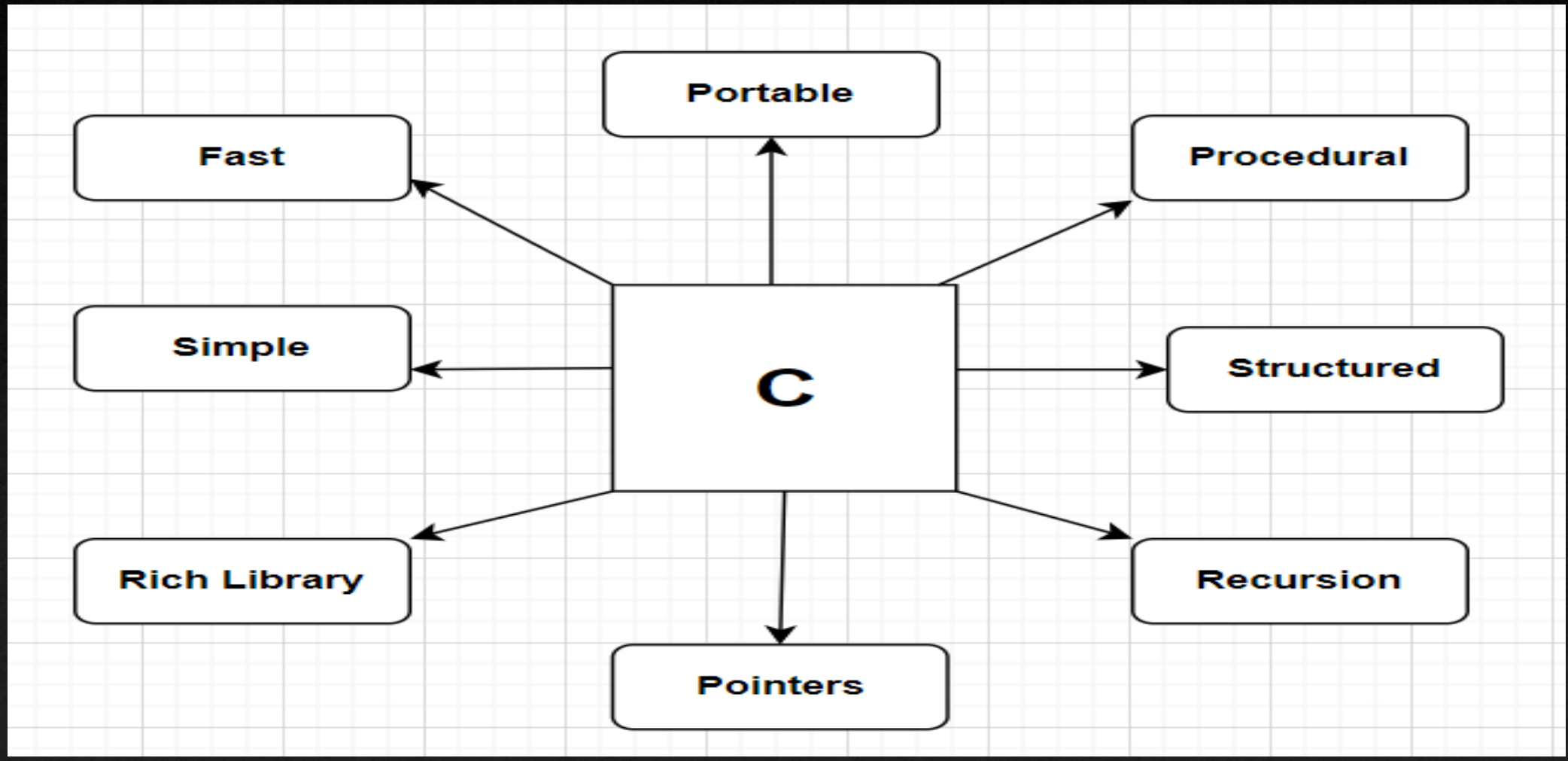
- Introduction to C
- Important features
- Get started with C
- First C Program
- Compilation Process
- Basic Syntax:
  - Identifier, Keywords
  - Variable, Data types, format specifier
  - Operators
  - Truth in C

Lesson 1 : C Basics

Introduction to C

- One of the oldest and most popular general purpose programming language.
- Developed in 1972 by Dennis Ritchie at bell laboratories of AT&T(American Telephone & Telegraph), located in U.S.A.
- Dennis Ritchie is known as founder of C language.
- Strongly associated with UNIX, as it was developed to write the UNIX operating system.

Features of C



Get Started With C

- Two things required to start C:
  1. A text editor like notepad to write the code
  2. A Compiler like GCC to translate the code to machine understandable code

- IDE (Integrated Development Environment) can also be used. Many popular IDEs are freely available either online or offline.
- Online IDEs like
  - https://onecompiler.com/c
  - https://www.onlinegdb.com/online_c_compiler
- Offline IDEs like
  - Visual Studio Code
  - Turbo C/C++ Compiler
  - Eclipse
  - NetBeans

1st Program in C

```c
#include<stdio.h>
// My first C program.
int main()
{
   printf("Hello World ! \n");
   return 0;
}
```

- First ensure, GCC is installed properly

```
C:\Users\Lenovo\day1> gcc --version
gcc (MinGW.org GCC-6.3.0-1) 6.3.0
Copyright (C) 2016 Free Software Foundation, Inc.
This is free software; see the source for copying
conditions.
```

- Compile & Run as shown below :

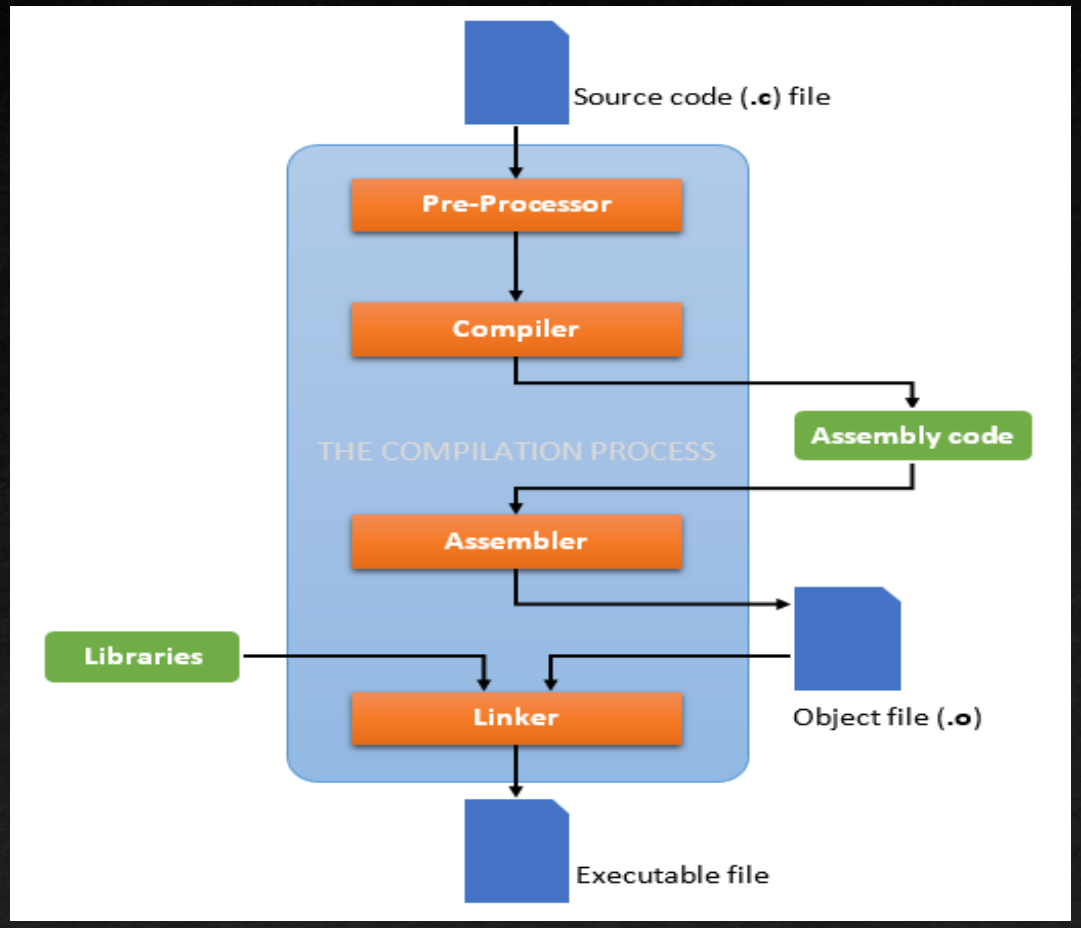
```
C:\Users\Lenovo\day1>gcc hello.c -o hello

C:\Users\Lenovo\day1>hello
Hello World !
```

Format of C program

- **#include <stdio.h>** includes the **standard input output** library functions. The printf() function is defined in stdio.h
- **main() function is the entry point of every program** in c language
- Statements are terminated with semicolons
- Indentation is ignored by the compiler
- C is case sensitive - all keywords and Standard Library functions are lowercase
- Strings are placed in double quotes
- Newlines are handled via \n
- Programs are capable of flagging success or error

Compilation Steps



1. Preprocessing
   gcc –E hello.c –o hello.i

2. Compiling
   gcc –S hello.c –o hello.s

3. Assembling
   gcc –c hello.c –o hello.o

4. Linking
   gcc hello.c –o hello

2nd Program in C

```c
#include<stdio.h>
// Calculate average of two numbers
int main()
{
    int a, b; float avg;
    printf("Enter two integer numbers \n");
    scanf("%d", &a);
    scanf("%d", &b);
   avg = (a + b) / 2.0;
   printf("Average of %d and %d is : %f", a, b, avg);
}
```

- `printf` *writes* values to screen
- `scanf` *reads* values from the keyboard
- a, b , avg are variables
- int, float are called data types
- %d is format specifier for integer

Basic Syntax : Identifiers

- A “C identifier” is a name used to identify a variable, function, or any other user-defined item.
- An identifier starts with a letter A to Z or a to z or an underscore _ followed by zero or more letters, underscores, and digits (0 to 9).
- C does not allow punctuation characters such as @, $, and % within identifiers.
- C is a **case sensitive** programming language.
- Thus, *Sum* and *sum* are two different identifiers in C. Here are some examples of acceptable identifiers:

mohd zara abc move_name a_123

myname50 _temp j a23b9 retVal

Basic Syntax : Keywords

- Keywords are reserved words which has special meaning in the language.
- Keywords can not be used as an identifier.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

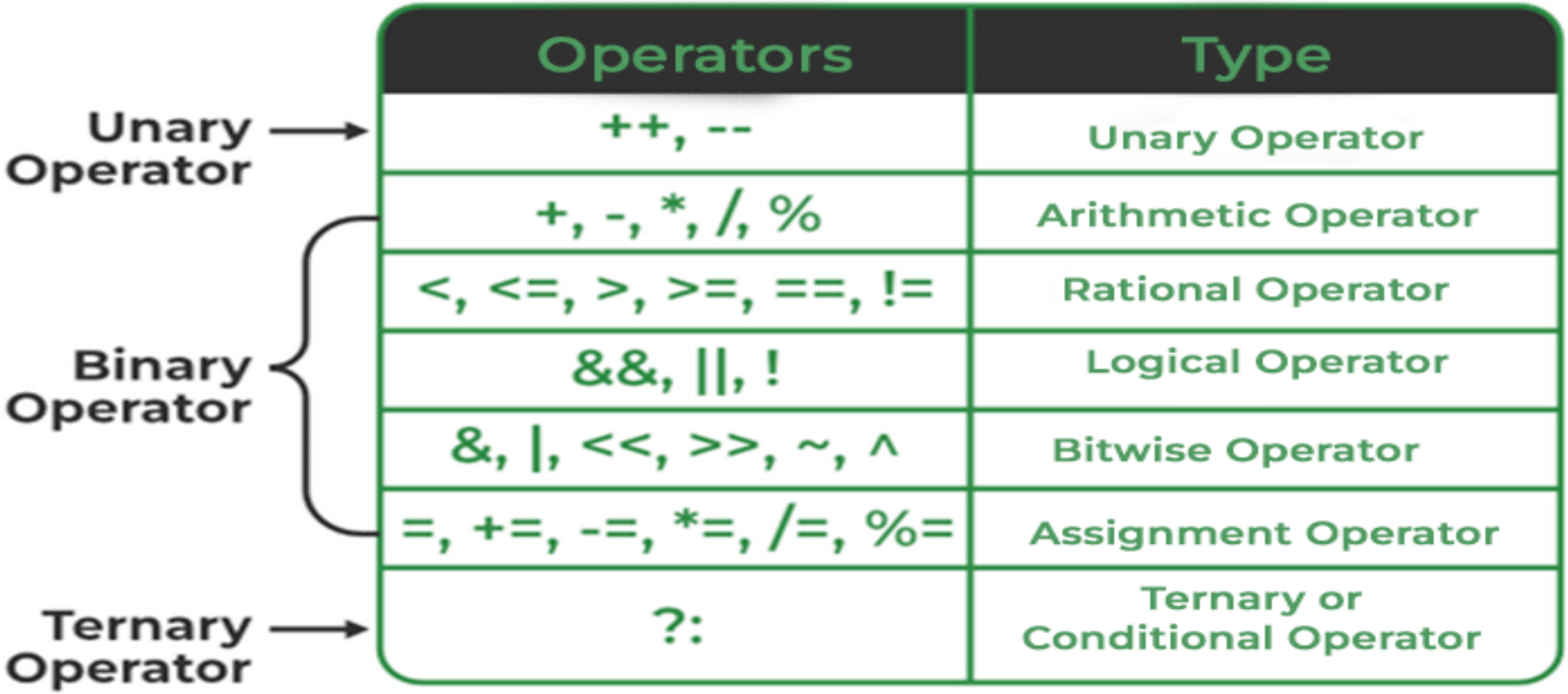
Variable, Data types & Format specifier

- Variables： are containers for storing data values, like numbers and characters.
- Date types：specifies the size and type of information the variable will store.
- Format specifier：is used to tell the compiler about the type of data to be printed or scanned in input and output operations. They always start with a **%** symbol and are used in the formatted string in functions like printf(), scanf

Basic built-in data types & format specifier

Data Type	Size(bytes)	Format Specifier	Description	Example
int	2 or 4	%d or %i	Whole numbers without decimals	10
float	4	%f or %F	Fractional numbers, containing one or more decimals(upto 7 decimal digits)	10.55
double	8	%lf	fractional numbers, containing one or more decimals(upto 15 decimal digits)	10.55
char	1	%c	single character/letter/number, or ASCII values	'A'

Operators



Operators	Type
++, --	Unary Operator
+, -, *, /, %	Arithmetic Operator
<, <=, >, >=, ==, !=	Rational Operator
&&, ||, !	Logical Operator
&, |, <<, >>, ~, ^	Bitwise Operator
=, +=, -=, *=, /=, %=	Assignment Operator
?:	Ternary or Conditional Operator

Truth in C

- To understand C's comparison operators (less than, greater than, etc.) and the logical operators (and, or, not) it is important to understand how C regards truth
- There is no boolean data type in C, integers are used instead
- The value of 0 (or 0.0) is false
- Any other value, 1, -1, 0.3, -20.8, is true

- Example:

```
if(0) printf("This will never be printed."); // false

if(100) printf("This will always be printed."); //true
```

Assignment

- Assignment is more flexible than might first appear
- An assigned value is always made available for subsequent use

```
int  i, j, k, l, m, n;

i = j = k = l = m = n = 22;

printf("%i\n", j = 93);
```

- Here, “n=22” happens first, this makes 22 available for assignment to “m”. Assigning 22 to “m” makes 22 available to assignment to “l” and so on…
- “j” is assigned 93, the 93 is then made available to printf for printing.

Lesson 1: Summary

Here is what we learned

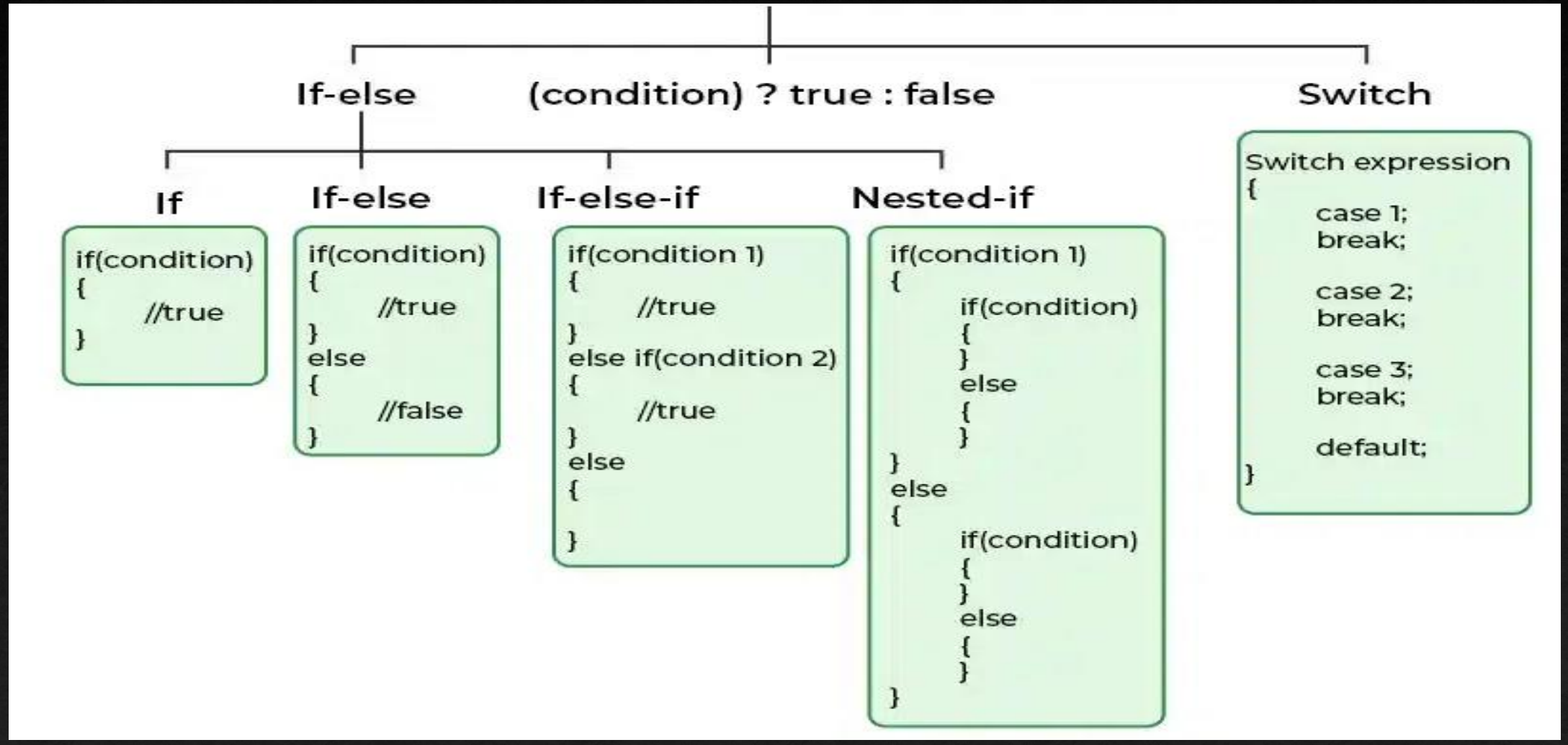
- What is C and its important features.
- How to write & run simple C programs.
- C basic constructs:
  - Keywords, Identifiers
  - Variables, Data types, format specifiers
  - Important functions
  - Operators
  - Truth in C
  - Assignments

Lesson 2 : Topics

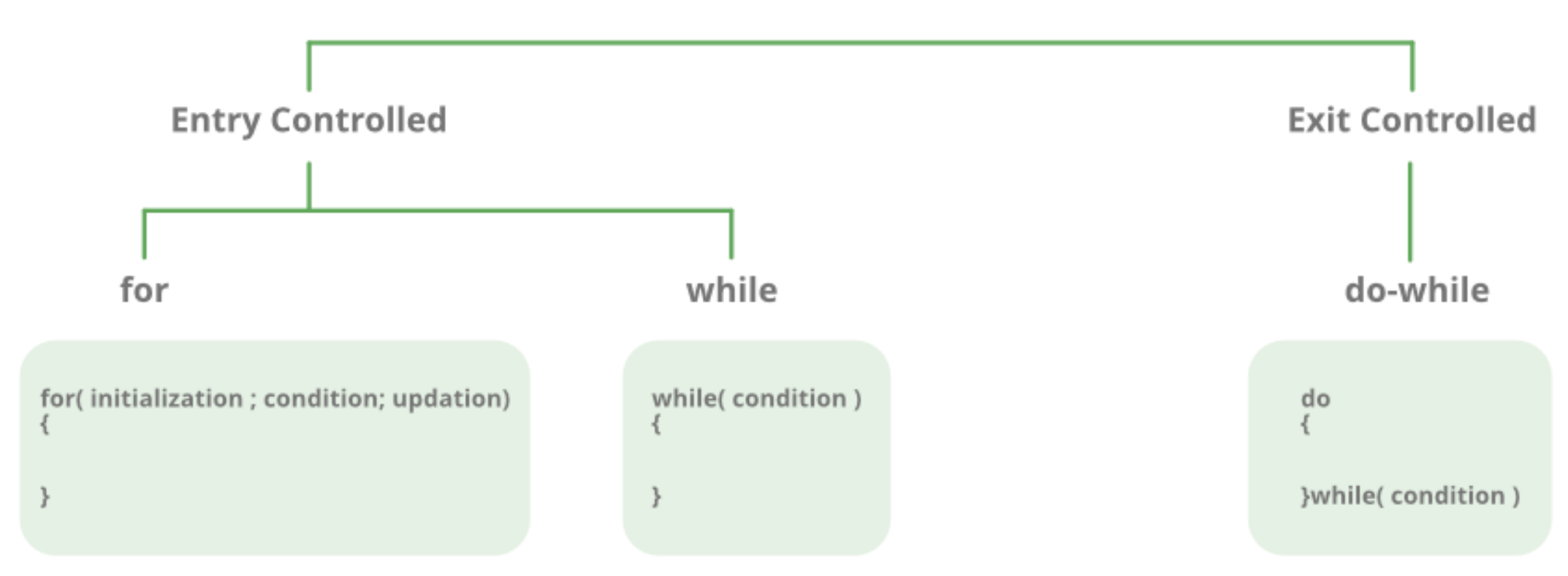
- If-else
- ternary operator
- switch
- while loop
- for loop
- do-while loop
- break, continue

Lesson 2 : Control Statements

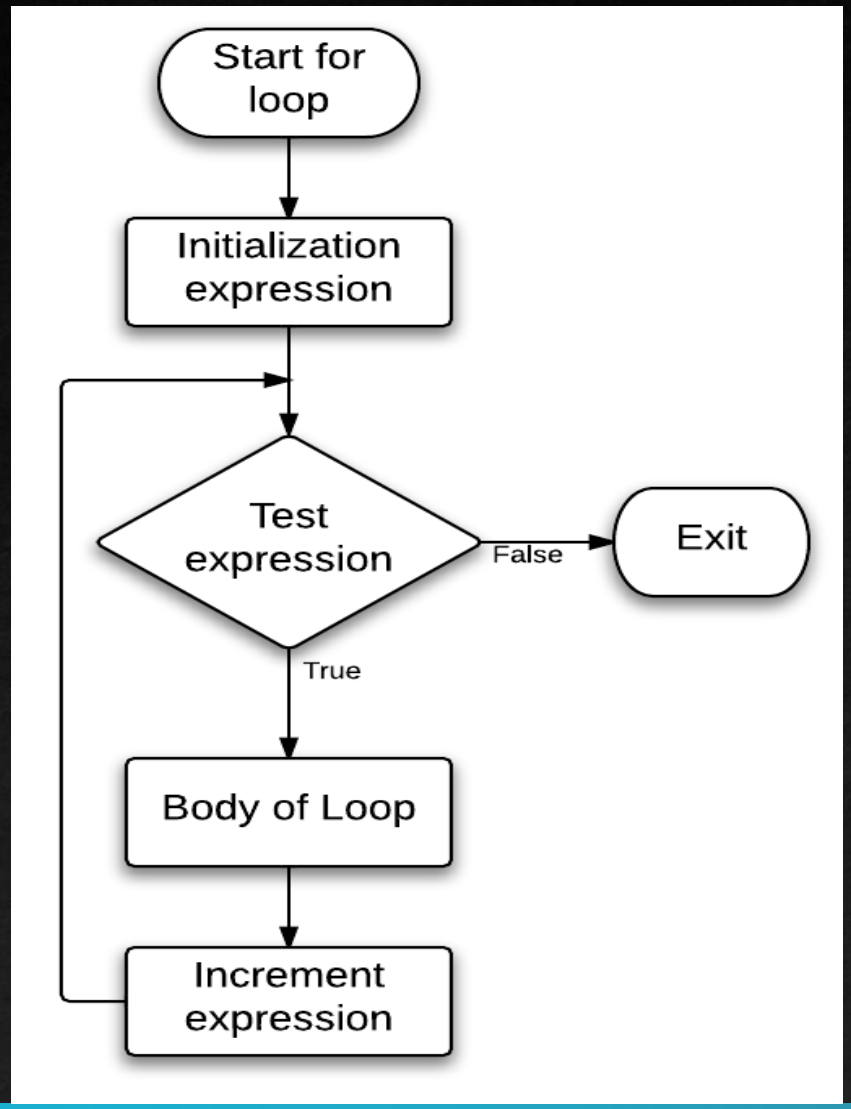
Control Statements : Conditions



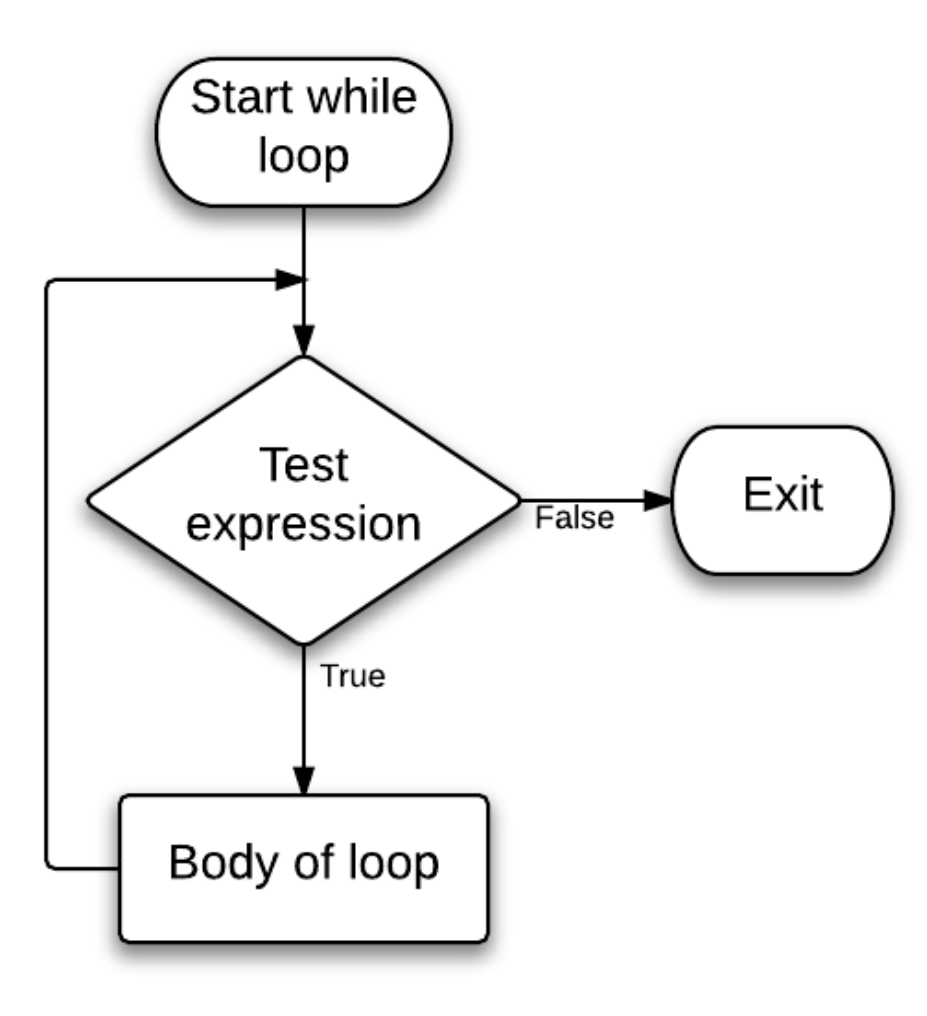
Control statements : Loops



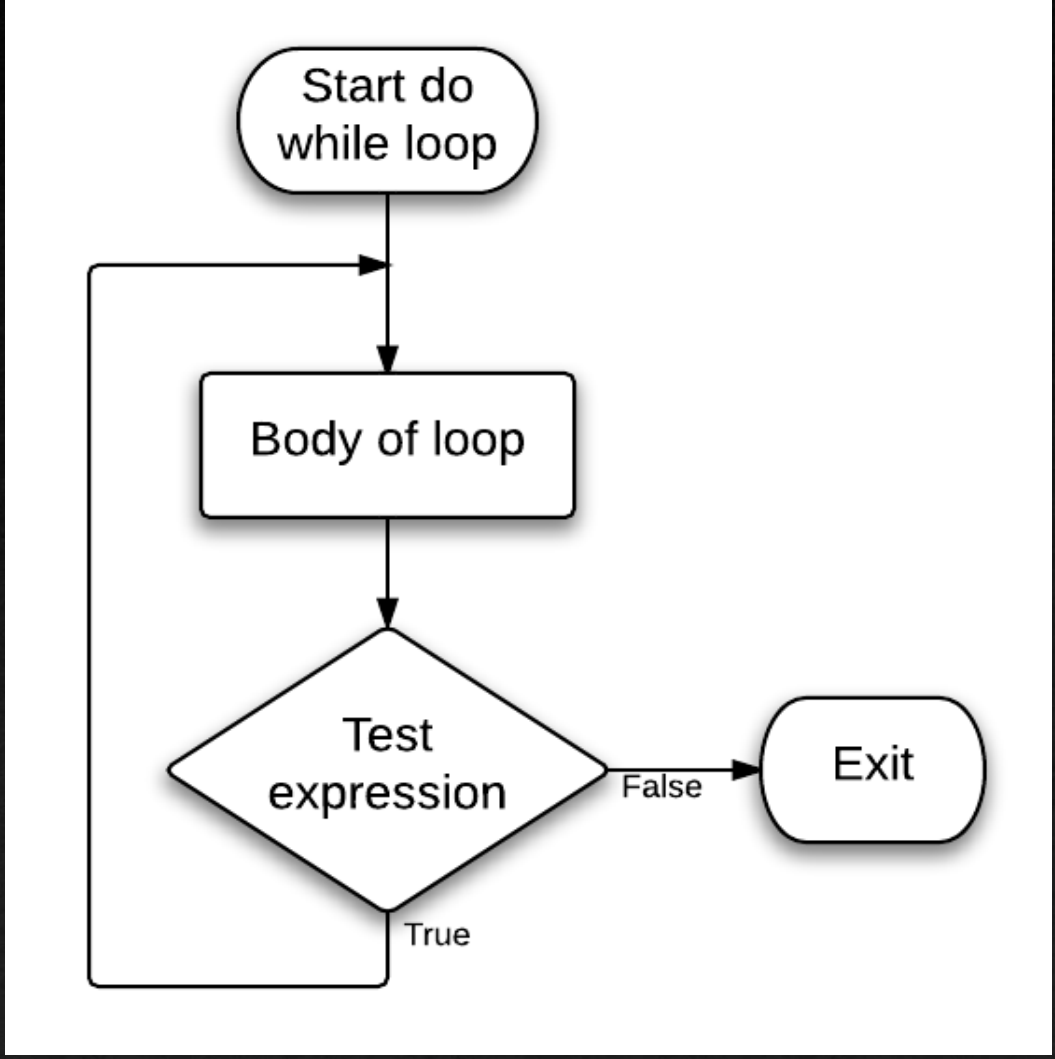
Loops : for loop



Loops : while loop



Loops : do-while loop



Lesson 2: Summary

Here is what we learned

- Control Statements
- If – Else
- Switch
- Ternary operator
- Loops - for, while, do-while
- break, continue

Lesson 3 : Topics

- Functions in C
- Function Syntax
- Pre defined functions
- Call by value
- Call by reference

Lesson 3 : Functions

Function in C

- A function is a block of code that performs a specific task.
- It has a name.
- It may accepts 1 or more parameters.
- It may or may not return value.
- It executes only when it is called.
- A function can be called multiple times.

- Advantages :
  - Code Reusability
  - Modularity

Function syntax in C

- The general form of a function definition in C programming language is as follows:

```
return_type  function_name( parameter list )
{
   body of the function
}
```

Example :

```
int  getSum(int x, int y)
{
   return x + y;
}
```

Pre defined Functions in C

- main() is a function which is the starting point to execute our code.
- printf(), scanf() are also functions.

Pass by value vs Pass by reference

- In pass by value, value being passed to the function is locally stored by the function parameter in stack memory location.
- If you change the value of function parameter, it is changed for the current function only.
- It will not change the value of variable inside the caller method such as main().

- In pass by reference, original value is modified because we pass reference (address).

Function call : pass by value

```c
#include <stdio.h>

void foo(int x)
{
    x = x * 10;
    printf("In foo, x = %d \n", x);
}

int main()
{
  int x = 100;
  printf("In main, x = %d \n", x);

  foo(x);

  printf("In main, x after calling foo = %d \n", x);
}
```

Function call : pass by reference

```c
#include <stdio.h>

void foo(int* x)
{
    *x = *x * 10;
    printf("In foo, x = %d \n", *x);
}

int main()
{
  int x = 100;
  printf("In main, x = %d \n", x);

  foo(&x);

  printf("In main, x after calling foo = %d \n", x);
}
```

Storage

- C stores local variables on the stack
- Global variables may be declared. These are not stack based, but are placed in the data segment
- Special keywords exist to specify where local variables are stored:

  `auto` - place on the stack (default)

  `static` - place in the data segment

  `register` - place in a CPU register

- Data may also be placed on the heap, this will be discussed later.

Lesson 3: Summary

Here is what we learned

- Functions in C
- Pass by value
- Pass by reference

Lesson 4 : Arrays

- Arrays in C
- Arrays Syntax
- Size of array
- Iterating array
- Passing array to function

Lesson 4 : Arrays

Arrays

- An array is a collection of data items (called *elements*) all of the same type
- It is declared using a type, a variable name and a CONSTANT placed in square brackets
- C always allocates the array in a single block of memory
- The size of the array, once declared, is fixed forever.

Creating array

```c
#include<stdio.h>

int main()
{
    //creating array of int of size 3 with values
   int x[] = {1,2,3};

   // accessing array elements by index
   printf("x[0] = %d \n", x[0]);

   // changing value
   x[0] = x[0] * 100;
   printf("x[0] = %d \n", x[0]);

   // declare array of int of size 2
   int y[2];

   // storing data
   y[0] = 901; y[1] = 902;

   // accessing array elements by index
   printf("y[0] = %d \n", y[0]);

   return 0;
}
```

Array size

- ```
  int myNumbers[] = {10, 25, 50, 75, 100};
  printf("%d", sizeof(myNumbers)); // Prints 20
  ```

- ```
  int myNumbers[] = {10, 25, 50, 75, 100};
  int length = sizeof(myNumbers) / sizeof(myNumbers[0]);

  printf("%d", length);  // Prints 5
  ```

Iterating array

```c
#include<stdio.h>

int main()
{
    int x[] = {1,2,3,4,5};
    int size = sizeof(x)/sizeof(x[0]);
    printf("array size=%d \n", size);

    // iterating array
    for(int i=0; i<size; i++)
    {
        printf("%d \n", x[i]);
    }

    return 0;
}
```

Passing array to function

```c
#include<stdio.h>

int get_sum(int y[], int size)
{
    int sum = 0;
    for(int i=0; i<size; i++)
    {
      sum = sum + y[i];
    }
    return sum;
}

int main()
{
    int x[] = {1,2,3,4,5};
    int size = sizeof(x)/sizeof(x[0]);

    int sum = get_sum(x, size);

    printf("sum of array elements = %d \n", sum);

    return 0;
}
```

Lesson 4: Summary

Here is what we learned

- Arrays in C

Lesson 5 : Strings

- Strings in C
- Example of String
- Size of string
- Iterating string char
- Passing string to function

Lesson 5 : Strings

Strings in C

- Strings are used to store text/characters.
- For example, "Hello World" is a string of characters.

- C has no native string type, instead we use arrays of `char`
- A special character, called a “null”, marks the end (don’t confuse this with the NULL pointer)
- This may be written as ‘\0’ (zero not capital ‘o’)
- This is the only character whose ASCII value is zero
- Depending on how arrays of characters are built, we may need to add the null by hand, or the compiler may add it for us

Creating String

```c
#include<stdio.h>

int main()
{
    char msg1[] = "Hello"; // termination char added by compiler

    char msg2[] = {'H','e','l','l','o','\0'};

    printf("size of msg1 = %d \n", sizeof(msg1));
    printf("size of msg2 = %d \n", sizeof(msg2));

    printf("msg1 = %s \n", msg1); //displays Hello
    printf("msg2 = %s \n", msg2); //display Hello

    return 0;
}
```

String functions

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

int main() {
    char msg[] = "Good Morning";
    printf("msg = %s \n", msg);
    printf("length of msg = %d \n", strlen(msg)); //12
    printf("sizeof of msg = %d \n", sizeof(msg)); //13

    char msg2[13];
    strcpy(msg2, msg); // copies msg to msg2
    printf("msg2 = %s \n", msg2);

    char msg3[50] = "Hi, ";
    strcat(msg3,msg); //concatenate msg in msg3
    printf("msg3 = %s \n", msg3);

    printf("Checking if msg & msg2 equals: \n");
    int result = strcmp(msg, msg2);
    result==0 ? printf("Equal \n") : printf("Not Equals \n");

    return 0;
}
```

Lesson 5: Summary

Here is what we learned

- Strings in C