

ICS-104

Introduction to C Notes

By: Naif Alqahtani

 Twitter: @NaifAlqahtani

 Youtube.com/MWNLLT

Beginning notes

- Unlike python, in C, you need to include a module (library) to use functions similar to `print()` & `input()` in python.
- The library you must include is the **standard input and output library** or **stdio** for short.
- To include libraries, write the following at the start of your file:

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

Diagram annotations for the code above:

- An arrow points to `#` with the label "not comment!".
- An arrow points to `include` with the label "keyword".
- An arrow points to `<` with the label "name".
- An arrow points to `stdio` with the label "name".
- An arrow points to `.h` with the label "extension".
- An arrow points to `>` with the label "name".

- In C comments are written using `//` for single line comments, or `/* ... */` for multiline comment.
- Unlike python, C is not sensitive to white spaces or indentation. However, it is good practice to indent your code correctly to make it neater.
- Since C does not rely on indentation, we use semi-colons `;` after every code statement that does not use curly brackets `{}`.
- Constants are defined using the prefix `#define` as so:


```
#define PI 3.14159
```
- `#define` constants are used to literally swap out the words in code for their value before the program is compiled. This means they cannot be modified elsewhere at all.
- `#include` and `#define` are called **preprocessor directives** since they are used to modify the code **before** it is compiled.

• The Main Function

- In C, we have a default function called **main** where the main code is placed. Every C program MUST have one.
- **main** returns integer values. It returns **0** if everything runs correctly and any other integer (other than zero) if an error needs to be raised. These return values are called exit codes. Basically, returning zero means success, any other number means failure

```
*include <stdio.h> // call library for printf
```

```

return type of function → int main(void)
function name →
void means no arguments →

Beginning of main → {
    printf("Hello!\n"); // print hello
    return 0; // return exit code
end of main → }
```

• Reserved keywords:

- int, double, float, char, void ...
- if, else, for, while, do, return ...

- Standard identifiers are key words that have a special meaning, but the programmer can redefine e.g: **printf, scanf**

- Therefore, if you created a function that you named **printf** then you can not use the one in C.

• User-defined identifiers:

- Cannot begin with numbers
- Letters, numbers, and underscores.
- No reserved names.

• Variables

- Same as in python, variables hold values that can be modified during runtime.
- However, unlike python, the variable **type** must be declared beforehand.
- You must declare any variable **before** calling/using it.
- You cannot modify a variable to another type after declaring it.

• Data Types

- **int** → stores whole integers only: 63, -123, 4002
- **double** → stores real numbers: 3.1415, 1.3e6 (1300000.0)
- **Float** → exactly like double but is less precise and uses less memory.
- **char** → stores exactly one character: 'a', '?', '3'

→ chars use single quotes.

→ chars store the ASCII code of each character.

"A" → "Z" : 65 → 90

"a" → "z" : 97 → 122

"0" → "9" : 48 → 57

" " : 32

"*" : 42

Input and Output

Python → C

print() → printf()

input() → scanf()

Both lines produce the exact same result

print("I m %d years old" % 20) ⇒ printf("I m %d years old\n", 20);

Diagram: Brackets above the code identify placeholders. In the Python code, brackets are under %d and %20, with an arrow pointing to the word 'Placeholder' above them. In the C code, brackets are under %d, \n, and 20, with an arrow pointing to the word 'Placeholder' above them.

You can use % Formatting in C too.

Types of place holders

Placeholder	Variable Type	Function Use
%c	char	printf/scanf
%d	int	printf/scanf
%f	double	printf
%lf	double	scanf

For input, we use scanf()

scanf works similar to printf. One difference is that we need to pass the address of the variable that stores the input.

To get the address of a variable we use & in front of the variable name.

Both set of lines are equivalent:

age = input("Enter age: ") ⇒ printf("Enter age: ");
scanf("%d", &age);

expect input of type int

get address of variable age

• Arithmetic Expressions

Arithmetic Operator	Meaning	Examples
+	addition	5 + 2 is 7 5.0 + 2.0 is 7.0
-	subtraction	5 - 2 is 3 5.0 - 2.0 is 3.0
*	multiplication	5 * 2 is 10 5.0 * 2.0 is 10.0
/	division	5.0 / 2.0 is 2.5 5 / 2 is 2
%	remainder	5 % 2 is 1

- The resulting types are as follows:

$\text{int} + \text{int} = \text{int}$

$\text{double} + \text{int} = \text{double}$

- When both operands are of type int , the result is also of type int .
- A mixed-type expression is an expression where one operand is of type int and the other is of type double .
- The result of a mixed-type expression will be of type double .

→ $5 + 5 = 10$

→ $5.0 + 5 = 10.0$

- When assigning a double type expression to a variable of type int , the expression will be floored and converted to type int . i.e the decimal places will be removed.

• $\text{int } y = 5/2$ → $y = 2$ ^{int}

• $\text{double } y = 5/2$ → $y = 2.5$ ^{double}

• $\text{int } y = 19/10$ → $y = 1$ ^{int}

- This is equivalent to $\text{int}(5.8) \rightarrow 5$ in python.

• Type Conversion (CASTS)

- To convert a value to another type:

$(\text{double})(9/2) \rightarrow 4.5$ ← double

$(\text{int})(9*0.5) \rightarrow 4$ ← int (floored to 4)

- Note: converting from double to int **Floors** the expression and does NOT round it to nearest integer

• Arithmetic Operators

- There are two types of operators in C:

- Unary \rightarrow needs only One operand (+, -, ~~X~~ only two)

- Binary \rightarrow needs two operands (+, -, *, /, %, etc)

- Unary operators are used to increment or decrement a variable by ONE.

$n++$ Unary
one operand \rightarrow is the same as $\rightarrow n = n + 1$ Binary
two operands

- The same applies to $n--$.

- Binary operators are the normal ones we used in python with a few exceptions.

- The placement of ++ or -- matters.

- In this case:

$n++$, the variable is used **THEN** incremented

- In this case:

$++n$, the variable is incremented **THEN** used.

IMPORTANT

Rules for Evaluating Expressions

- Parentheses are always evaluated FIRST.
- Inside parentheses are evaluated before outside ones.
- Order of operators:
 - First: Unary Operators $+$, $-$
 - Second: $*$, $/$, $\%$
 - Third: Binary Operators $+$, $-$
- Unary Operators in the same subexpression and same precedence level are evaluated Right to Left
- Binary Operators in the same subexpression and same precedence level are evaluated Left to Right
- Some Arithmetic operations in python can be used in C:
 - $+=$ $-=$ $*=$ $/=$
 - $+$ $-$ $/$ $\%$
- The power operator cannot be used in C (**).

Condition Statements (Selection)

C:

```

    Required
    if (x > y)
    {
        Required
        x ++ ;
        y = 3 ;
    }
    notice else if
    else if (x == y)
    {
        Required
        x = 3 + 2 ;
        y = 0 ;
    }
    else
    {
        Required
        x -- ;
    }
    NOT Required
    but highly
    recommended
  
```

Python:

```


    Required
    if x > y :
        Required
        x = x + 1
        y = 3
    elif x == y :
        Required
        x = 3 + 2
        y = 0
    else:
        Required
        x -= 1
  
```

- We cannot use and in C. Instead we use &&.
- We cannot use or in C. Instead we use ||.
- We cannot use not in C. Instead we use !.
- We can use:
 - < , >
 - <= , >=
 - ==
 - !=

• Logic Gate Truth Table

A	B	(A && B)	(A B)	!A
true	true	True	True	False
true	false	False	True	False
false	true	False	True	True
false	false	False	False	True

- In C: There is no boolean type (you need a library)
- Instead 0 is considered False and 1 is considered True

Operator	Precedence
function calls	highest
! + - & (unary operators)	
* / %	
+ -	
< <= >= >	
== !=	
&& (logical AND)	
(logical OR)	lowest
= (assignment operator)	

• Loops

Python:

```
while x > y:
    x += 1
    y -= 1
```

Annotations: "Required" with arrows pointing to the condition `x > y` and the loop body lines.

C:

```
while (x > y)
{
    x++;
    y--;
}
```

Annotations: "Required" with arrows pointing to the condition `(x > y)` and the opening curly brace. "Required if body is more than one line" with an arrow pointing to the opening curly brace. "Required" with an arrow pointing to the increment `x++`. "NOT Required but highly recommended" with an arrow pointing to the closing curly brace.

Python:

```
for x in range(5):
    print(x)
```

Annotations: "Required" with arrows pointing to the condition `range(5)` and the loop body line.

C:

```
for (int i=0 ; i<=5 ; i++)
{
    printf("%d", i);
}
```

Annotations: "Required" with arrows pointing to the initialization `i=0`, the condition `i<=5`, and the increment `i++`. "Required if body is more than one line" with an arrow pointing to the opening curly brace. "NOT Required but highly recommended" with an arrow pointing to the closing curly brace.

• The For loop

Diagram illustrating the components of a C-style For loop:

```
For (int i=0 ; i<5 ; i++)
```

Annotations:

- int i=0**: initialize loop variables
- i<5**: repeat loop while this condition is true
- i++**: do this after completing every loop
- ;**: separate each statement by semi-colon

- Nested For loops work in theory works exactly as in python. Please revise my Chapte 4 notes.

• do... while

- The do...while statement works exactly like while
- The difference being that the body of the while loop must be executed at least **ONCE** regardless if the condition is met or not.

```
do
{
    x ++ ;
    y -- ;
} while ( x > y );
```

Required if body is more than one line

NOT Required but highly recommended

Required

Required