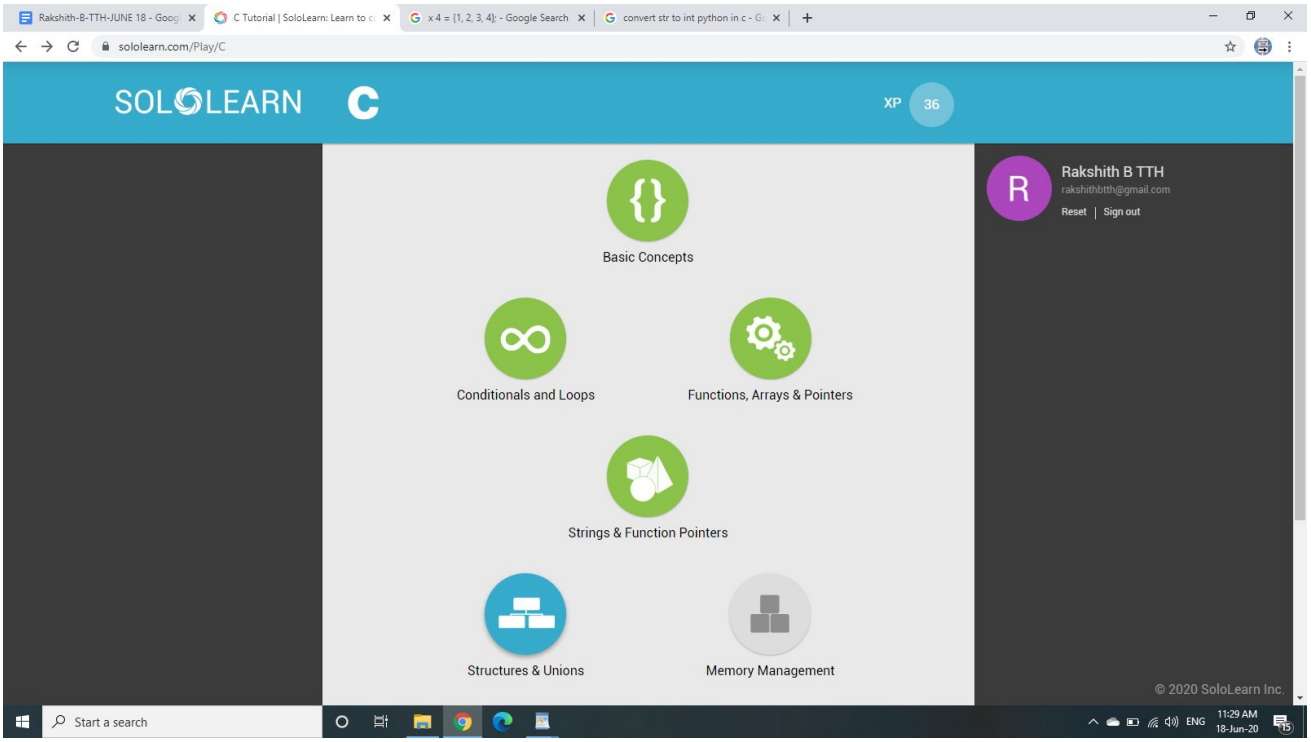
REPORT JUNE 18

Date:	18 JUNE 2020	Name:	Rakshith B
Course:	Solo Learning	USN:	4AL16EC409
Topic:	Basic Concepts,Conditionals and Loops,Functions,Arrays & Pointers,Strings and Function Pointers	Semester & Section:	6th SEM B
Github Repository:	Rakshith-B		

FORENOON SESSION DETAILS

Image of session



Report –

Introducing C

C is a general-purpose programming language that has been around for nearly 50 years.

C has been used to write everything from operating systems (including Windows and many others) to complex programs like the Python interpreter, Git, Oracle database, and more.

The versatility of C is by design. It is a low-level language that relates closely to the way machines work while still being easy to learn.

Hello World!

Let's break down the code to understand each line:

#include <stdio.h> The function used for generating output is defined in stdio.h. In order to use the printf function, we need to first include the required file, also called a header file.

int main() The main() function is the entry point to a program. Curly brackets { } indicate the beginning and end of a function (also called a code block). The statements inside the brackets determine what the function does when executed.

```
#include <stdio.h>

int main() {
 printf("Hello, World!\n");
 return 0;
}
```

Data Types

C supports the following basic data types:

int: integer, a whole number.

float: floating point, a number with a fractional part.

double: double-precision floating point value.

char: single character.

The amount of storage required for each of these types varies by platform.

C has a built-in sizeof operator that gives the memory requirements for a particular data type.

Variables

A variable is a name for an area in memory.

The name of a variable (also called the identifier) must begin with either a letter or an underscore and can be composed of letters, digits, and the underscore character.

Variable naming conventions differ, however using lowercase letters with an underscore to separate words is common (snake_case).

Variables must also be declared as a data type before they are used.

The value for a declared variable is changed with an assignment statement.

For example, the following statements declare an integer variable my_var and then assigns it the value 42

```
#include <stdio.h>
int main() {
 int a, b;
```

```
 float salary = 56.23;
 char letter = 'Z';
 a = 8;
 b = 34;
 int c = a+b;
 printf("%d \n", c);
 printf("%f \n", salary);
 printf("%c \n", letter);
 return 0;
}
```

Constants

A constant stores a value that cannot be changed from its initial assignment.
By using constants with meaningful names, code is easier to read and understand.
To distinguish constants from variables, a common practice is to use uppercase identifiers.

One way to define a constant is by using the const keyword in a variable declaration:

```
#include <stdio.h>

int main() {
 const double PI = 3.14;
 printf("%f", PI);

 return 0;
}
```

Input

C supports a number of ways for taking user input.
getchar() Returns the value of the next single character input.

For example:

```
#include <stdio.h>

int main() {
 char a = getchar();

 printf("You entered: %c", a);

 return 0;
}
```

The input is stored in the variable a.

The gets() function is used to read input as an ordered sequence of characters, also called a string.

Formatted Input

The scanf() function is used to assign input to variables. A call to this function scans input according to format specifiers that convert input as necessary.
If input can't be converted, then the assignment isn't made.
The scanf() statement waits for input and then makes assignments:

```
int x;
float num;
char text[20];
scanf("%d %f %s", &x, &num, text);
```

Typing 10 22.5 abcd and then pressing Enter assigns 10 to x, 22.5 to num, and abcd to text.
Note that the & must be used to access the variable addresses. The & isn't needed for a string because a string name acts as a pointer.

Comments

Comments are explanatory information that you can include in a program to benefit the reader of your code. The compiler ignores comments, so they have no affect on a program.
A comment starts with a slash asterisk /* and ends with an asterisk slash */ and can be anywhere in your code.
Comments can be on the same line as a statement, or they can span several lines.
For example:

```
#include <stdio.h>

/* A simple C program
*  Version 1.0
*/
int main() {
 /* Output a string */
 printf("Hello World!");
 return 0;
}
```

Single-line Comments

C++ introduced a double slash comment // as a way to comment single lines. Some C compilers also support this comment style.
For example:

```
#include <stdio.h>

int main() {
 int x = 42; //int for a whole number

 //%d is replaced by x
 printf("%d", x);

 return 0;
}
```

Arithmetic Operators

C supports arithmetic operators + (addition), - (subtraction), * (multiplication), / (division), and % (modulus division).

Operators are often used to form a numeric expression such as 10 + 5, which in this case contains two operands and the addition operator.
Numeric expressions are often used in assignment statements.
For example:

```
#include <stdio.h>

int main() {
 int length = 10;
 int width = 5;
 int area;

 area = length * width;
 printf("%d \n", area);  /* 50 */

 return 0;
}
```

Operator Precedence

C evaluates a numeric expression based on operator precedence.
The + and – are equal in precedence, as are *, /, and %.
The *, /, and % are performed first in order from left to right and then + and -, also in order from left to right.
You can change the order of operations by using parentheses ( ) to indicate which operations are to be performed first.
For example, the result of 5 + 3 * 2 is 11, where the result of (5 + 3) * 2 is 16.

For example:

```
#include <stdio.h>

int main() {
 int a = 6;
 int b = 4;
 int c = 2;
 int result;
 result = a - b + c; // 4
 printf("%d \n", result);

 result = a + b / c; // 8
 printf("%d \n", result);

 result = (a + b) / c; // 5
 printf("%d \n", result);

 return 0;
}
```

Conditionals

Conditionals are used to perform different computations or actions depending on whether a condition evaluates to true or false.

The if Statement

The if statement is called a conditional control structure because it executes statements when an expression is true. For this reason, the if is also known as a decision structure. It takes the form:

```
if (expression)
 statements
```

The expression evaluates to either true or false, and statements can be a single statement or a code block enclosed by curly braces { }.

For example:

```
#include <stdio.h>

int main() {
 int score = 89;

 if (score > 75)
   printf("You passed.\n");

 return 0;
}
```

Relational Operators

There are six relational operators that can be used to form a Boolean expression, which returns true or false:

< less than
<= less than or equal to
> greater than
>= greater than or equal to
== equal to
!= not equal to

For example:

```
int num = 41;
num += 1;
if (num == 42) {
  printf("You won!");
}
```

The if-else Statement

The if statement can include an optional else clause that executes statements when an expression is false.

For example, the following program evaluates the expression and then executes the else clause statement:

```
#include <stdio.h>

int main() {
 int score = 89;

 if (score >= 90)
   printf("Top 10%%.\n");
 else
   printf("Less than 90.\n");

 return 0;
}
```

Conditional Expressions

Another way to form an if-else statement is by using the ?: operator in a conditional expression. The ?: operator can have only one statement associated with the if and the else.

For example:

```
#include <stdio.h>

int main() {
 int y;
 int x = 3;

 y = (x >= 5) ?  5 : x;

/* This is equivalent to:
 if (x >= 5)
   y = 5;
 else
   y = x;
*/

 return 0;
}
```

The switch Statement

The switch statement branches program control by matching the result of an expression with a constant case value.

The switch statement often provides a more elegant solution to if-else if and nested if statements.

The switch takes the form:

```
switch (expression) {
```

```
 case val1:
   statements
 break;
 case val2:
   statements
 break;
 default:
   statements
}
```

Logical Operators

Logical operators && and || are used to form a compound Boolean expression that tests multiple conditions. A third logical operator is ! used to reverse the state of a Boolean expression.

The && Operator

The logical AND operator && returns a true result only when both expressions are true.

For example:

```
if (n > 0 && n <= 100)

 printf("Range (1 - 100).\n");
```

The while Loop

The while statement is called a loop structure because it executes statements repeatedly while an expression is true, looping over and over again. It takes the form:

```
while (expression) {
 statements
}
```

The expression evaluates to either true or false, and statements can be a single statement or, more commonly, a code block enclosed by curly braces { }.

Functions in C

Functions are central to C programming and are used to accomplish a program solution as a series of subtasks.

By now you know that every C program contains a main() function. And you're familiar with the printf() function.

You can also create your own functions.
A function:
- **is a block of code that performs a specific task**
- **is reusable**
- **makes a program easier to test**
- **can be modified without changing the calling program**

Even a simple program is easier to understand when main() is broken down into subtasks that are implemented with functions.
For example, it's clear that the goal of this program is to calculate the square of a number:

```
int main() {
 int x, result;

 x = 5;
 result = square(x);
 printf("%d squared is %d\n", x, result);

 return 0;
}
```

Recursive Functions

An algorithm for solving a problem may be best implemented using a process called recursion.
Consider the factorial of a number, which is commonly written as 5! = 5 * 4 * 3 * 2 * 1.
This calculation can also be thought of as repeatedly calculating num * (num -1) until num is 1.

A recursive function is one that calls itself and includes a base case, or exit condition, for ending the recursive calls. In the case of computing a factorial, the base case is num equal to 1.
For example:

```
#include <stdio.h>

//function declaration
int factorial(int num);

int main() {
 int x = 5;
 printf("The factorial of %d is %d\n", x, factorial(x));
  return 0;
}
//function definition
int factorial(int num) {
 if (num == 1)  /* base case */
  return (1);
 else
```

```
  return (num * factorial(num - 1));
}
```