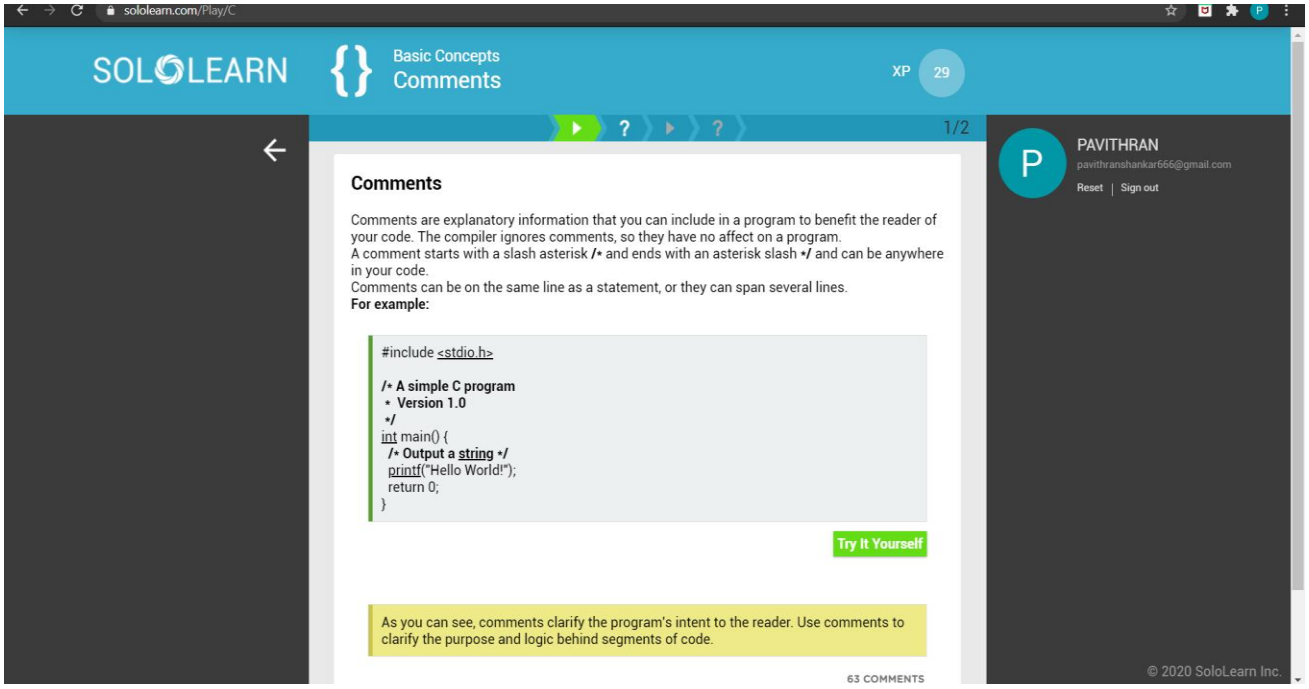
DAILY ASSESSMENT FORMAT

Date:	**19 JUNE 2020**	**Name:**	**PAVITHRAN S**
Course:	**C PROGRAMMING**	**USN:**	**4AL17EC068**
Topic:	**BASICS**	**Semester & Section:**	**6TH B**
Github Repository:	**Pavithran**		

FORENOON SESSION DETAILS

Image of session



Report – Report can be typed or hand written for up to two pages.

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;
}
```

Nested if Statements

An **if** statement can include another **if** statement to form a nested statement. Nesting an **if** allows a decision to be based on further requirements.
Consider the following statement:

```
if (profit > 1000)
if (clients > 15)
```

```
bonus = 100;
else
bonus = 25;
```

The if-else if Statement

When a decision among three or more actions is needed, the **if-else if** statement can be used.
There can be multiple **else if** clauses and the last **else** clause is optional.
For example:

```
int score = 89;

if (score >= 90)
printf("%s", "Top 10%\n");
else if (score >= 80)
printf("%s", "Top 20%\n");
else if (score > 75)
printf("%s", "You passed.\n");
else
printf("%s", "You did not pass.\n");
```

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
}
```

For example, the following program outputs "Three":

```
int num = 3;

switch (num) {
case 1:
```

```
printf("One\n");
break;
case 2:
printf("Two\n");
break;
case 3:
printf("Three\n");
break;
default:
printf("Not 1, 2, or 3.\n");
}
```

The switch Statement

There can be multiple **cases** with unique labels.
The optional **default** case is executed when no other matches are made.
A **break** statement is needed in each case to branch to the end of the **switch** statement.
Without the **break** statement, program execution falls through to the next **case** statement.
This can be useful when the same statement is needed for several cases. Consider the following **switch** statement:

```
switch (num) {
case 1:
case 2:
case 3:
printf("One, Two, or Three.\n");
break;
case 4:
case 5:
case 6:
printf("Four, Five, or Six.\n");
break;
default:
printf("Greater than Six.\n");
}
```

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 || Operator

The logical OR operator **||** returns a true result when any one expression or both expressions are true.
For example:

```
if (n == 'x' || n == 'X')
printf("Roman numeral value 10.\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 { }.
For example:

```
#include <stdio.h>

int main() {
int count = 1;

while (count < 8) {
printf("Count = %d\n", count);
count++;
}

return 0;
}
```

The do-while Loop

The **do-while** loop executes the loop statements before evaluating the expression to determine if the loop should be repeated.
It takes the form:do {
statements
} while (expression);
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 count = 1;

do {
printf("Count = %d\n", count);
count++;
} while (count < 8);

return 0;
}
```

break and continue

The **break** statement was introduced for use in the **switch** statement. It is also useful for immediately exiting a loop.
For example, the following program uses a **break** to exit a **while** loop:

```
int num = 5;

while (num > 0) {
if (num == 3)
break;
printf("%d\n", num);
num--;
}
```

The for Loop

The **for** loop can contain multiple expressions separated by commas in each part.
For example:for (x = 0, y = num; x < y; i++, y--) {

```
statements;
}
```
Also, you can skip the **initvalue**, **condition** and/or **increment**.
For example:
```
int i=0;
int max = 10;
for (; i < max; i++) {
printf("%d\n", i);
}
```