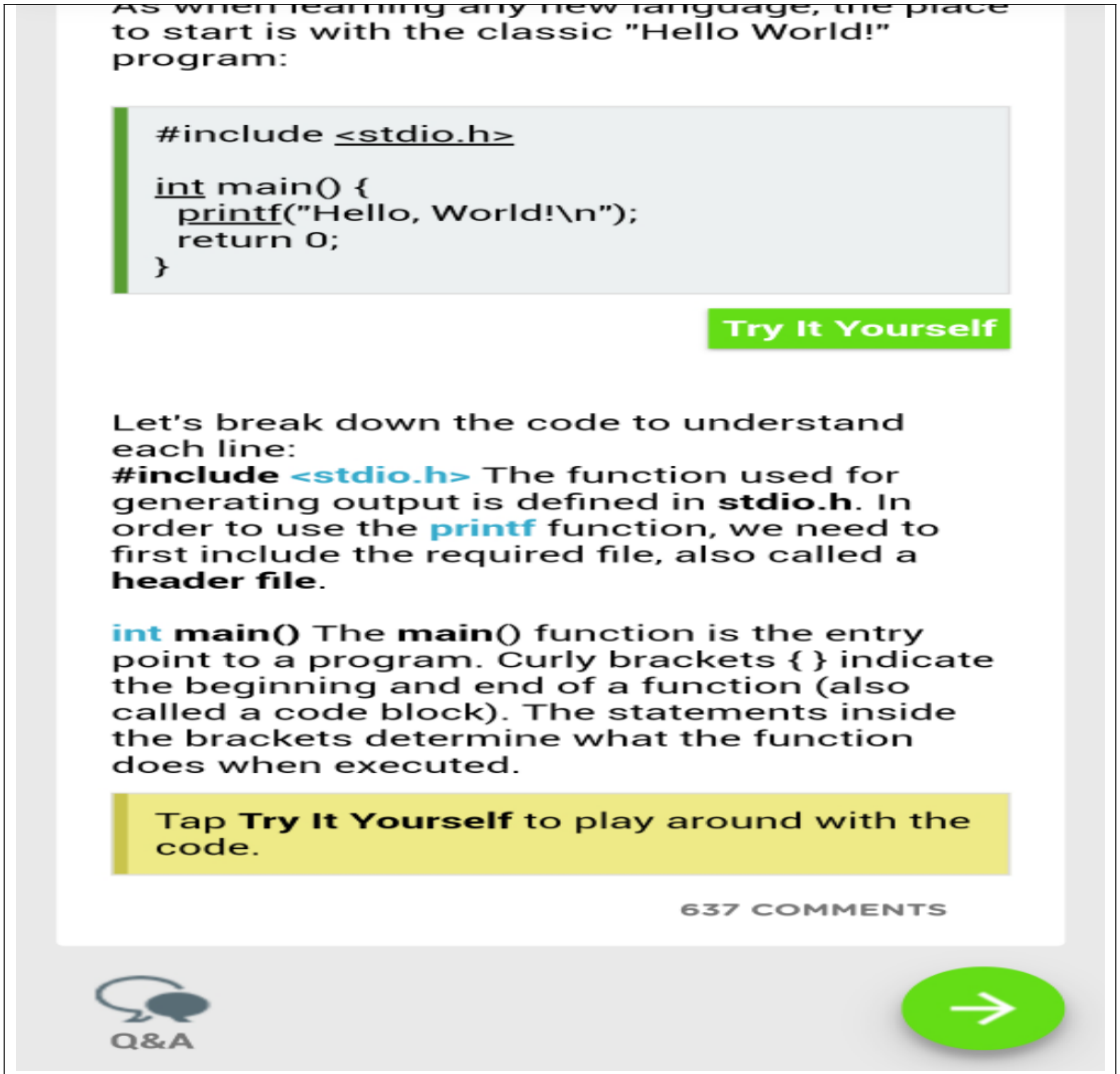
DAILY ASSESSMENT FORMAT

Date:	19-06-2020	Name:	Jagadeesha Hegde
Course:	C programming	USN:	4AL17EC036
Topic:	Structures & Unions	Semester & Section:	6th A-sec
Github Repository:	Jagadeesha-036		

FORENOON SESSION DETAILS
Image of session



As when learning any new language, the place to start is with the classic "Hello World!" program:

```
#include <stdio.h>

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

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.

Tap **Try It Yourself** to play around with the code.

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

structures in c :structure is a user-defined data type available in C that allows to combining data items of different kinds. Structures are used to represent a record. Defining a structure: To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than or equal to one member. The format of the struct statement is as follows:

```
struct [structure name]

{

 member definition;

 member definition;

 ...

 member definition;

};
```

UNION:

A union is a special data type available in C that allows storing different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple purposes.

Defining a Union: To define a union, you must use the union statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows:C is a procedural programming language. It was initially developed by Dennis Ritchie as a system programming language to write operating system. The main features of C language include low-level access to memory, simple set of keywords, and clean style, these features make C language suitable for system programming like operating system or compiler development.

```
union [union name]

{ member definition;

 member definition;
...
member definition;
};
```

Similarities between Structure and Union

1. Both are user-defined data types used to store data of different types as a single unit.

2. Their members can be objects of any type, including other structures and unions or arrays. A member can also consist of a bit field.

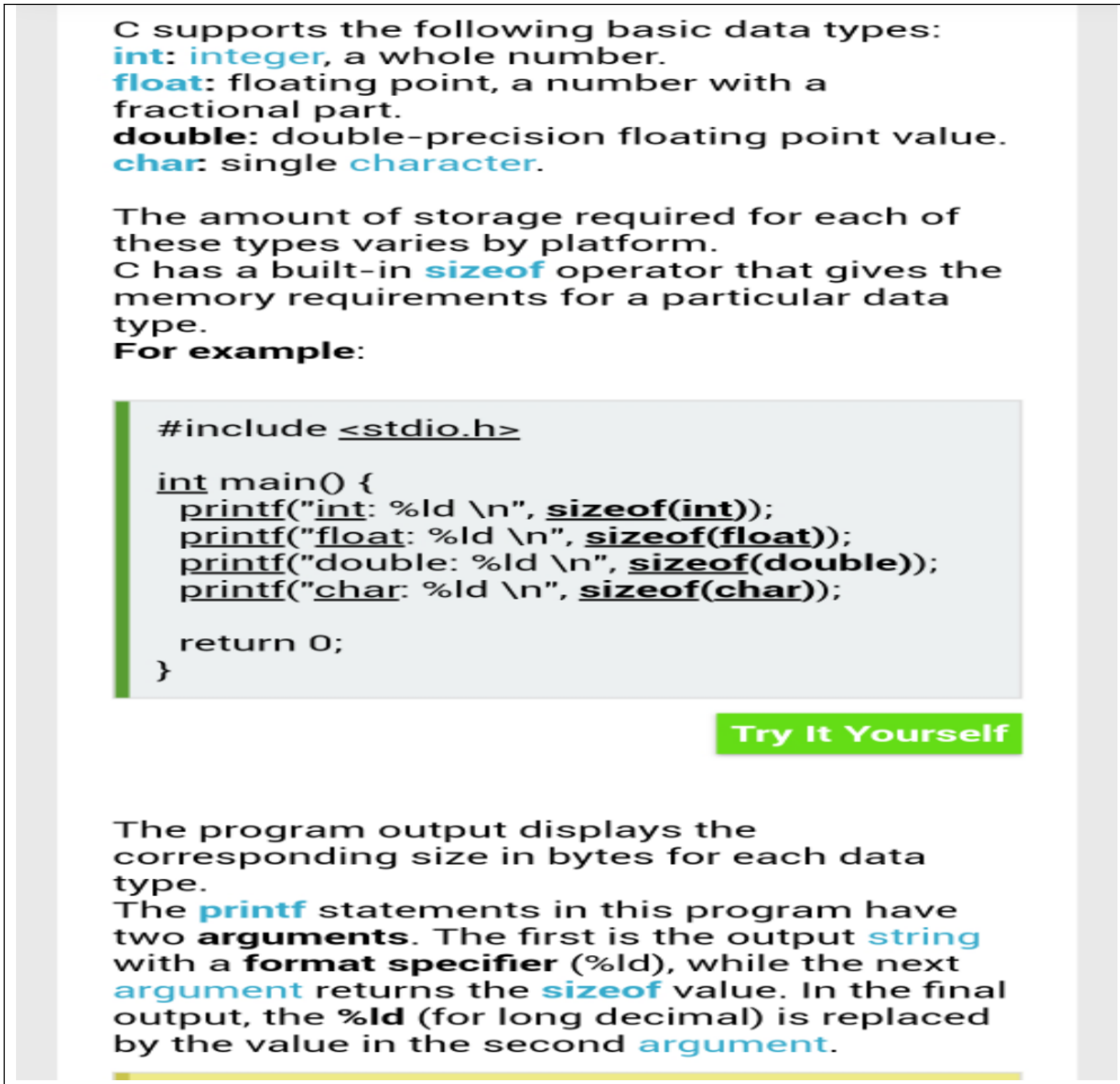
3. Both structures and unions support only assignment = and sizeof operators. The two structures or unions in the assignment must have the same members and member types.

4. A structure or a union can be passed by value to functions and returned by value by functions. The argument must have the same type as the function parameter. A structure or union is passed by value just like a scalar variable as a corresponding parameter.

5. '.' operator is used for accessing members.

Date:	19-06-2020	Name:	Jagadeesha Hegde
Course:	C programming	USN:	4AL17EC036
Topic:	Memory Management	Semester & Section:	6th A-sec

AFTERNOON SESSION DETAILS
Image of session



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.
For example:

```
#include <stdio.h>

int main() {
  printf("int: %ld \n", sizeof(int));
  printf("float: %ld \n", sizeof(float));
  printf("double: %ld \n", sizeof(double));
  printf("char: %ld \n", sizeof(char));

  return 0;
}
```

The program output displays the corresponding size in bytes for each data type.
The **printf** statements in this program have two **arguments**. The first is the output string with a **format specifier** (%ld), while the next argument returns the **sizeof** value. In the final output, the **%ld** (for long decimal) is replaced by the value in the second argument.

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

Memory Management in C :

When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default. One of the things that makes C such a versatile language is that the programmer can scale down a program to run with a very small amount of memory. When C was first written, this was an important feature because computers weren't nearly as powerful as they are today. With the current demand for small electronics, from mobile phones to tiny medical devices, there's a renewed interest in keeping the memory requirements small for some software. C is the go-to language for most programmers who need a lot of control over memory usage.To better understand the importance of memory management, consider how a program uses memory. When you first run a program, it loads into your computer's memory and begins to execute by sending and receiving instructions from the computer's processor. When the program needs to run a particular function, it loads that function into yet another part of memory for the duration of its run, then abandons that memory when the function is complete. Plus, each new piece of data used in the main program takes up memory for the duration of the program.

There are two ways in which memory can be allocated in C:

• by declaring variables

• by explicitly requesting space from C

We have discussed variable declaration in other lectures, but here we will describe requesting dynamic memory allocation and memory management.

C provides several functions for memory allocation and management:

• malloc and calloc, to reserve space

• realloc, to move a reserved block of memory to another allocation of different dimensions

• free, to release space back to C

These functions can be found in the stdlib library

What happens when a pointer is declared?

Whenever a pointer is declared, all that happens is that C allocates space for the pointer.

For example,

```
char *p;
```

allocates 4 consecutive bytes in memory which are associated with the variable p. p's type is declared to be of pointer to char. However, the memory location occupied by p is not initialised, so it may contain garbage.

It is often a good idea to initialise the pointer at the time it is declared, to reduce the chances of a random value in p to be used as a memory address:

char *p = NULL;

At some stage during your program you may wish p to point to the location of some string A common error is to simply copy the required string into p: strcpy(p, "Hello");

Often, this will result in a "Segmentation Fault". Worse yet, the copy may actually succeed.

```
//a.c


#include <stdio.h>

main()

{

     char *p;

      char *q = NULL;

     printf("Address of p = %u\n", p);

      strcpy(p, "Hello");

     printf("%s\n", p);

     printf("About to copy \"Goodbye\" to q\n");

     strcpy(q, "Goodbye");

    printf("String copied\n");

    printf("%s\n", q);

}
```

When p and q are declared, their memory locations contain garbage. However, the garbage value in p happens to correspond to a memory location that is not write protected by another process. So the strcpy is permitted. By initialising q to NULL, we are ensuring that we cannot use q incorrectly. Trying to copy the string "Goodbye" into location 0 (NULL) results in a run-time Bus Error, and a program crash