

DAILY ASSESSMENT FORMAT

Date:	19/06/2020	Name:	Akshatha M Deshpande
Course:	SoloLearn	USN:	4AL17EC006
Topic:	C Programming	Semester & Section:	6th Sem A sec
Github Repository:	AkshathaDeshpande		

FORENOON SESSION DETAILS

Image of session

The screenshot displays the SoloLearn C programming playground. The interface includes a top navigation bar with 'COURSES', 'CODE PLAYGROUND' (highlighted), 'DISCUSS', 'TOP LEARNERS', 'BLOG', and 'MY CODES(1)'. Below the navigation bar, there are tabs for 'c', 'Dark', 'Light', 'C', and 'Output'. The 'c' tab is selected, showing the following C code:

```
1 #include <stdio.h>
2 #include <string.h>
3
4 struct course {
5     int id;
6     char title[40];
7     float hours;
8 };
9
10 int main() {
11     struct course cs1 = {341279, "Intro to C++", 12.5};
12     struct course cs2;
13
14     /* Initialize cs2 */
15     cs2.id = 341281;
16     strcpy(cs2.title, "Advanced C++");
17     cs2.hours = 14.25;
18
19     /* display course info */
20     printf("%d\t%s\t%.2f\n", cs1.id, cs1.title, cs1.hours);
21     printf("%d\t%s\t%.2f\n", cs2.id, cs2.title, cs2.hours);
22
23     return 0;
24 }
```

The 'Output' tab shows the following output:

```
341279 Intro to C++ 12.50
341281 Advanced C++ 14.25
```

At the bottom of the interface, there are buttons for 'SAVE', 'SAVE AS', 'RESET', and 'RUN' (highlighted). Additionally, there are links to 'GET IT ON Google play' and 'Download on the App Store'.

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

Module 5:

A structure :

- A structure is a user-defined data type that groups related variables of different data types.
- A structure declaration includes the keyword struct, a structure tag for referencing the structure, and curly braces { } with a list of variable declarations called members.
- Do not forget to put a semicolon after structure declaration.
- A structure is also called a composite or aggregate data type. Some languages refer to structures as records.
- A struct variable is stored in a contiguous block of memory. The sizeof operator must be used to get the number of bytes needed for a struct, just as with the basic data types.
- The typedef keyword creates a type definition that simplifies code and makes a program easier to read.

Pointers to Structures:

- Just like pointers to variables, pointers to structures can also be defined.
- `struct myStruct *struct_ptr;`
defines a pointer to the myStruct structure.
- `struct_ptr = &struct_var;`
stores the address of the structure variable struct_var in the pointer struct_ptr.
- `struct_ptr -> struct_mem;`
accesses the value of the structure member struct_mem.



Structures as Function Parameters:

- A function can have structure parameters that accept arguments by value when a copy of the structure variable is all that is needed.
- For a function to change the actual values in a struct variable, pointer parameters are required.

Array of Structures:

- An array can store elements of any data type, including structures.
- After declaring an array of structures, an element is accessible with the index number.
- The dot operator is then used to access members of the element

Union:

- A union allows to store different data types in the same memory location.
- It is like a structure because it has members.
- However, a union variable uses the same memory location for all its member's and only one member at a time can occupy the memory location.

Pointers to Unions:

- A pointer to a union points to the memory location allocated to the union.
- A union pointer is declared by using the keyword union and the union tag along with * and the pointer name.

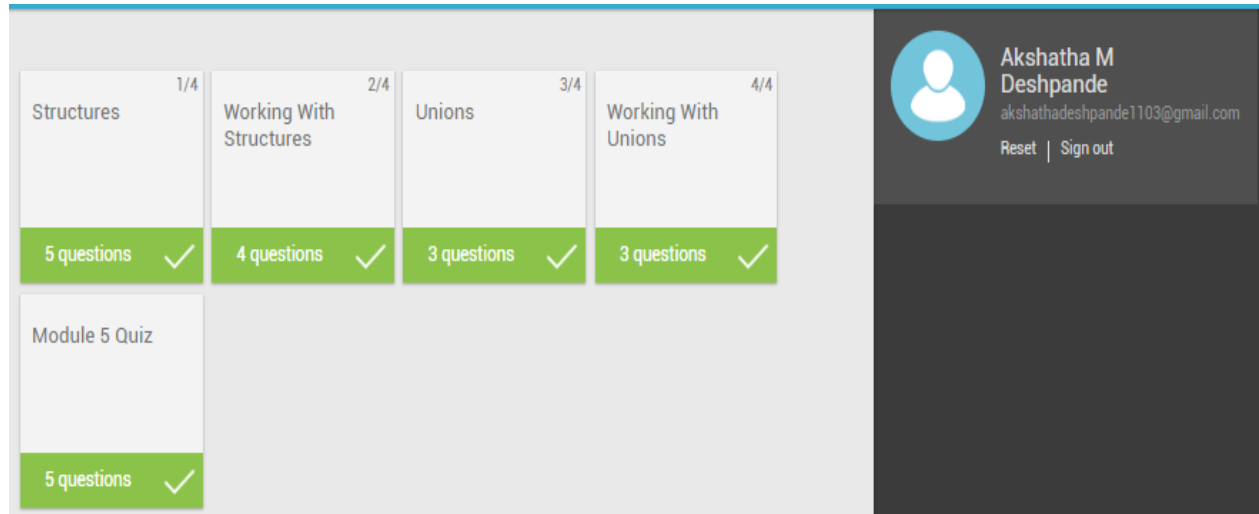
Unions as Function Parameters:

- A function can have union parameters that accept arguments by value when a copy of the union variable is all that is needed.
- For a function to change the actual value in a union memory location, pointer parameters are required.

Array of Unions:

- An array can store elements of any data type, including unions.
- With unions, it is important to keep in mind that only one member of the union can store data for each array element.





MODULE 6:

Memory Management:

- Understanding memory is an important aspect of C programming. When you declare a variable using a basic data type, C automatically allocates space for the variable in an area of memory called the stack.
- An int variable, for example, is typically allocated 4 bytes when declared. We know this by using the sizeof operator
- Dynamic memory allocation is the process of allocating and freeing memory as needed. Now you can prompt at runtime for the number of array elements and then create an array with that many elements.
- Dynamic memory is managed with pointers that point to newly allocated blocks of memory in an area called the heap.
- The **stdlib.h** library includes memory management functions.
- The statement **#include <stdlib.h>** at the top of your program .

The malloc Function:

- The malloc() function allocates a specified number of contiguous bytes in memory.

- The allocated memory is contiguous and can be treated as an array. Instead of using brackets [] to refer to elements, pointer arithmetic is used to traverse the array. You are advised to use + to refer to array elements. Using ++ or += changes the address stored by the pointer.
- If the allocation is unsuccessful, NULL is returned. Because of this, you should include code to check for a NULL pointer.

The calloc Function:

- The calloc() function allocates memory based on the size of a specific item, such as a structure.
- The program below uses calloc to allocate memory for a structure and malloc to allocate memory for the string within the structure

The realloc Function:

- The realloc() function expands a current block to include additional memory.
- realloc leaves the original content in memory and expands the block to allow for more storage.

Allocating Memory for Strings:

- When allocating memory for a string pointer, you may want to use string length rather than the sizeof operator for calculating bytes.
- When using strlen to determine the number of bytes needed for a string, be sure to include one extra byte for the NULL character '\0'.
- A char is always one byte, so there is no need to multiply the memory requirements by sizeof(char).

Dynamic Arrays:

- Many algorithms implement a dynamic array because this allows the number of elements to grow as needed.
- Because elements are not allocated all at once, dynamic arrays typically use a structure to keep track of current array size, current capacity, and a pointer to the elements, as in the following program.
- The entire program is written in main() for demonstration purposes.



- To properly implement a dynamic array, sub-tasks should be broken down into functions such as `init_array()`, `increase_array()`, `add_element()`, and `display_array()`. The error checking was also skipped to keep the demo short.


Working With Memory 1/4
2 questions ✓

The malloc Function 2/4
3 questions ✓

calloc and realloc 3/4
2 questions ✓

Dynamic Strings & Arrays 4/4
2 questions ✓

Module 6 Quiz
5 questions ✓



Akshatha M Deshpande
akshathadeshpande1103@gmail.com
Reset | Sign out

Date:	19/06/2020	Name:	Akshatha M Deshpande
Course:	Webinar by cambridge	USN:	4AL17EC006
Topic:	Negotiation skills	Semester & Section:	6th Sem A sec


Cambridge Assessment English









Empathy

Picture source: www.adefactgroup.com



