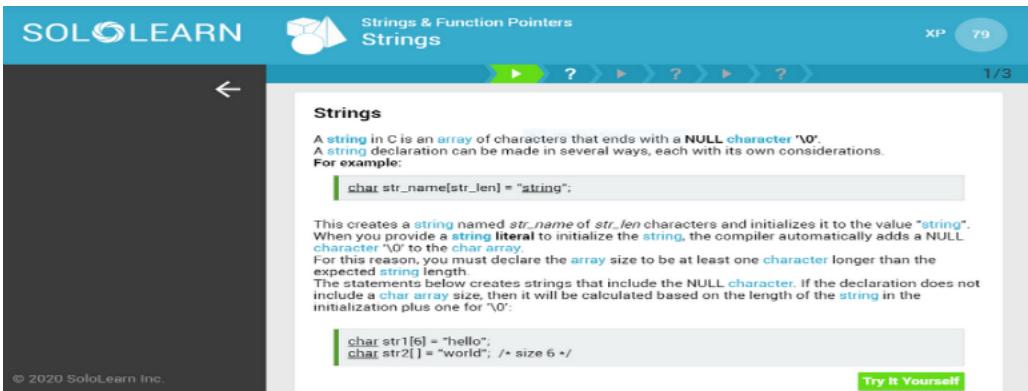
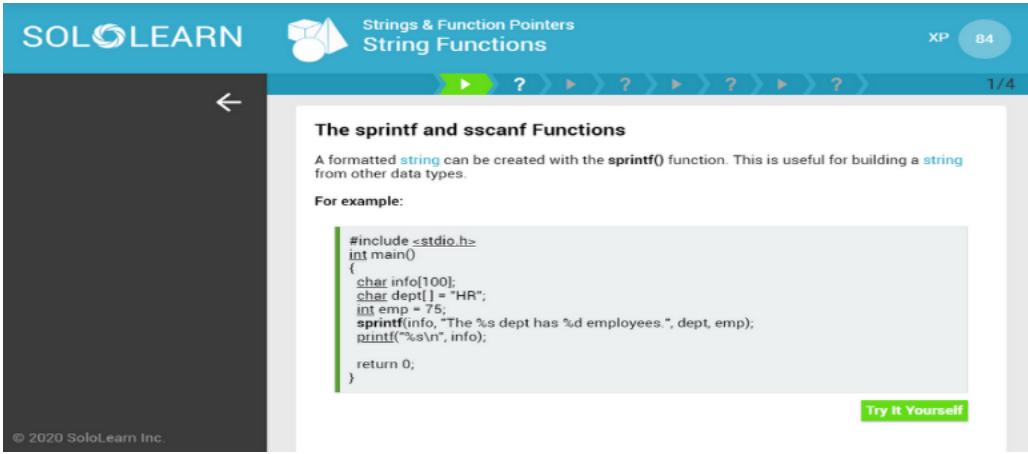


DAILY ASSESSMENT FORMAT

Date:	19-06-2020	Name:	Rajeshwari Gadagi
Course:	Solo Learn	USN:	4AL17EC076
Topic:	C Programming	Semester & Section:	6 SEM & 'B' SEC
Github Repository:	Rajeshwari-gadagi		

FORENOON SESSION DETAILS	
Image of session	 <p>The screenshot shows a mobile application interface for the SoloLearn platform. At the top, it says "SOLOLEARN" and "Strings & Function Pointers". Below that is a navigation bar with arrows and question marks. On the right, there's an "XP" icon with the number "79" and a progress bar showing "1/3". The main content area is titled "Strings". It contains text explaining that a string in C is an array of characters ending with a NULL character '\0'. It provides examples of string declarations:</p> <pre>char str_name[str_len] = "string";</pre> <p>This creates a string named str_name of str_len characters and initializes it to the value "string". When you provide a string literal to initialize the string, the compiler automatically adds a NULL character '\0' to the char array.</p> <p>For example:</p> <pre>char str1[6] = "hello"; char str2[] = "world"; /* size 6 */</pre> <p>At the bottom right is a green "Try It Yourself" button.</p>  <p>The screenshot shows another mobile application interface for the SoloLearn platform. At the top, it says "SOLOLEARN" and "String Functions". Below that is a navigation bar with arrows and question marks. On the right, there's an "XP" icon with the number "84" and a progress bar showing "1/4". The main content area is titled "The sprintf and sscanf Functions". It contains text explaining that a formatted string can be created with the sprintf() function. It provides an example:</p> <pre>#include <stdio.h> int main() { char info[100]; char dept[] = "HR"; int emp = 75; sprintf(info, "The %s dept has %d employees.", dept, emp); printf("%s\n", info); }</pre> <p>At the bottom right is a green "Try It Yourself" button.</p>

SOLOLEARN Structures & Unions Structures XP 101 1/5

Structures

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**.

For example:

```
struct course {  
    int id;  
    char title[40];  
    float hours;  
};
```

This struct statement defines a new data type named **course** that has three members. Structure members can be of any data type, including basic types, strings, arrays, pointers, and even other structures, as you will learn in a later lesson.

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**.

© 2020 SoloLearn Inc.

SOLOLEARN Structures & Unions Structures XP 101 2/5

Declarations Using Structures

To declare **variables** of a **structure** data type, you use the keyword **struct** followed by the **struct tag**, and then the **variable name**. For example, the statements below declares a **structure** data type and then uses the **student** **struct** to declare variables **s1** and **s2**.

```
struct student {  
    int age;  
    int grade;  
    char name[40];  
};  
  
/* declare two variables */  
struct student s1;  
struct student s2;
```

Try It Yourself

A **struct** variable is stored in a contiguous block of memory. The **sizeof** operator must be

© 2020 SoloLearn Inc.

C Programming

1st year - Friday

Strings & Function Pointers

> Strings

A string in C is an array of characters that ends with a null character (\0). A string declaration can be made in different ways, each with its own considerations. For example:

char str_name[10] = "Harry";

String pointer declarations declare above as `str = "Harry";`

`strlen()` - get length of a string

`strcmp()` - compare two strings

`strcpy()` - copy one string to another

`strncpy()` - copy string to maximum

`strncpy()` - convert string to upper case

`strrev()` - reverse string

`strcmp()` - compare two strings

> Getting Input:

Programs need user interaction, asking the user for input.

To do this we use `scanf()` to read input according to the format specifier.

A date (datatype) is `get()`, `scanf()`, which reads input in a specified no. of characters.

> Putting Output

Output aspect is handled with `printf()`, `put()` & `printff()` functions.

`printf()` requires the name of a pointer variable you want to print and where to print to the screen, use `std::cout` which refers to standard output.

> Getting Functions

The `printf()` & `scanf()` functions

A formatted string can be created with the `printf()` function. This is useful in building a string from other variable types.

Another useful function is `scanf()` for reading a string of values. The function reads values from a user & stores them at the corresponding variable addresses.

> The Standard Library

The standard library contains numerous string functions.

> Converting String to Number

Converting a string to no. of characters to numeric value is a common task in c programming. It often needs to implement user-defined classes.

> Arrays of Storage

A one-dimensional array can be used to store related storage.

Function Pointers

A function pointer can point to an address in memory. It's like a variable that points to another variable (function name / parameter).

* Arrays & Function Pointers

An array of function pointers can replace a switch statement for choosing an action, as in following:

Dereferencing pointers

A deref operator is used to refer to memory address defined in a memory of some declaration that looks like `int *ptr;`

* Functions Using void Pointers

Void pointers are often used in function declarations:

`void *square (const void *);`

* Function pointers as arguments

A function pointer used as an argument is sometimes referred to as a callback function, because the receiving function "calls it back".
`void square (void *box, size_t num, size_t width, int (*compute) (const void *, const void *));`

> Structures & Union Structures & Unions

A structure is a user-defined data type that groups related variables of different data types.

A structure declaration includes the keyword `struct`.

* Declaration Using Structure

To declare a variable of a structure data type, you use the keyword `struct` followed by the struct tag & then the variable name.

* Declarations Using Structures

A struct variable can also be initialized with the declaration by listing initial values in curly braces.

* Accessing Structure Members

You can access the members of a structure variable by using a dot operator (.) followed by the variable name & the member name.

* Using decltype

The type tag `decltype` creates a type definition that implicitly holds from a previous declaration to end.

> Working with Structures

* Structures with Other Data

The members of a structure may also be structures.

* Pointers to Structures

```
struct myStruct *struct_ptr;
struct_ptr = &struct_var;
struct_ptr -> struct_member;
```

- Structures as Function Parameters
A function can have structure parameters that accept arguments by value. When a copy of the structure formula is all that is needed.
- Array of Structures:
An array can store elements of any data types, including structures.
- > Unions
 - Union allows to store different data types in the same memory location. A union declaration uses the designated union, enclosing, curly braces {}, with a list of members.
 - Accessing Union members:
We can access the members of a union's variable by using the dot operator between the variable name & the member's name.
 - Structures with Unions:
Unions are often used with structures, because a structure can have members to keep track of which union member stored a value.
- > Pointers to Unions
 - Pointers to a union point to a memory location allocated to the union.
 - Union pointer is declared by using the tag word union & the union tag along with & of the pointer name.



Date:	19-06-2020	Name:	Rajeshwari Gadagi
Course:	Solo Learn	USN:	4AL17EC076
Topic:	C Programming	Semester & Section:	6 SEM & 'B' SEC
Github Repository:	Rajeshwari-gadagi		

AFTERNOON SESSION DETAILS

Image of session

Memory Management Functions

The `stdlib.h` library includes memory management functions. The statement `#include <stdlib.h>` at the top of your program gives you access to the following:

- `malloc(bytes)`** Returns a `pointer` to a contiguous block of memory that is of size `bytes`.
- `calloc(num_items, item_size)`** Returns a `pointer` to a contiguous block of memory that has `num_items` items, each of size `item_size` bytes. Typically used for arrays, structures, and other derived data types. The allocated memory is initialized to 0.
- `realloc(ptr, bytes)`** Resizes the memory pointed to by `ptr` to size `bytes`. The newly allocated memory is not initialized.
- `free(ptr)`** Releases the block of memory pointed to by `ptr`.

When you no longer need a block of allocated memory, use the function `free()` to make the block available to be allocated again.

44 COMMENTS 112 XP

The malloc Function

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

For example:

```
#include <stdlib.h>
int *ptr;
/* a block of 10 ints */
ptr = malloc(10 * sizeof(*ptr));
if (ptr != NULL) {
    *(ptr + 2) = 50; /* assign 50 to third int */
}
```

malloc returns a pointer to the allocated memory. Notice that `sizeof` was applied to `ptr` instead of `int`, making the code more robust should the `*ptr` declaration be changed to a different data type later.

Try It Yourself 117 XP



The realloc Function

The `realloc()` function expands a current block to include additional memory.
For example:

```
int *ptr;
ptr = malloc(10 * sizeof(*ptr));
if (ptr != NULL) {
    *(ptr + 2) = 50; /* assign 50 to third int */
}
ptr = realloc(ptr, 100 * sizeof(*ptr));
*(ptr + 30) = 75;
```

Try It Yourself

`realloc` leaves the original content in memory and expands the block to allow for more storage.

Tap Try It Yourself to play around with the code.

60 COMMENTS

C Programming

Memory Management

Understanding memory is an important aspect of C programming.
On the example, for example let's say we allocated 10 bytes when initialized. We know this by using the `sizeof` operator.

```
int x;
printf("%d", sizeof(x)); /* output: 4 */
```

* Memory Management Functions

The standard library includes memory management functions.

`#include <stdlib.h>`

`malloc()`

`calloc (num_items, item_size)`

`realloc (ptr, bytes)`

`free (ptr)`

> The malloc Function

The `malloc()` function allocates a specified number of contiguous bytes in a memory.
`malloc` return a pointer to the allocated memory.

* The malloc Function

The allocated memory is contiguous & can be treated as an array.

The free Function

The `free()` function is a memory management function that is called to release memory.
By freeing memory, you make it available for use later in your program.

2. calloc and realloc

The `calloc()` function allocates memory based on the size of a specific item, such as an array.

* realloc () Function

The `realloc()` function expands a current block to include additional memory.

> Allocating Memory of strange

when allocating memory after a string pointer, you may want to use a length greater than the `sizeof` operator for calculating bytes.

* Dynamic arrays

Many algorithms implement a dynamic array because this allows the no. of elements to grow or shrink.

SOLOLEARN

Courses Code Playground Discuss Top Learners Blog My Codes(39)

Dark Light C Output SHARE ↗

```
1 #include <stdio.h>
2
3 int main() {
4     char first_name[25];
5     int age;
6     printf("Enter your first name and age: \n");
7     scanf("%s %d", first_name, &age);
8
9     printf("\nHi, %s. Your age is %d", first_name, age);
10
11    return 0;
12 }
```

string_input_scanf Public

SAVE SAVE AS RESET ▶ RUN

Activate Windows Go to [GET IT ON Google play](#) Download on the [App Store](#)

This screenshot shows a C code editor on the SoloLearn platform. The code reads a first name and age from the user, then prints a greeting. The output window shows the result for 'reena' at age 18.

SOLOLEARN

Courses Code Playground Discuss Top Learners Blog My Codes(48)

Dark Light C Output SHARE ↗

```
1 #include <stdio.h>
2
3 int main() {
4     int x = 33;
5     float y = 12.4;
6     char c = 'a';
7     void *ptr;
8
9     ptr = &x;
10    printf("void ptr points to %d\n", *((int *)ptr));
11    ptr = &y;
12    printf("void ptr points to %f\n", *((float *)ptr));
13    ptr = &c;
14    printf("void ptr points to %c", *((char *)ptr));
15
16    return 0;
17 }
```

Activate Windows

This screenshot shows a C code editor on the SoloLearn platform. The code demonstrates how a void pointer can be cast to point to different types of variables (int, float, char) and then printed. The output window shows the results for each conversion.

