

Login



## Q

# C Language CheatSheet

#### **Basics**

Basic syntax and functions from the C programming language.

#### **Boilerplate Code**

## printf function

It is used to show output on the screen

```
printf("Hello World!");
```

#### scanf function

It is used to take input from the user

```
scanf("format_specifier", &variables)
example-
int a;
scanf("%d",&a);
printf("%d",&a);
```

#### Comments

A comment is the code that is not executed by the compiler, and the programmer uses it to keep track of the code.

## Single line comment

```
// It's a single line comment
```

#### Multi-line comment

```
/* It's a
multi-line
comment
*/
```

## **Data types**

The data type is the type of data

### Character type

Typically a single octet(one byte). It is an character type

```
char variable_name;
```

format specifier of character is "%c"

```
char x;
scanf(" %c",&x);
printf("character is %c",x)
```

## Integer type

To store non-decimal numeric values integer type is used

```
int variable_name;
```

format specifier of integer is "%d"

```
int a;
scanf("%d",&a);
printf("%d",a);
```

## Float type

To store decimal numeric values float type is used

```
float variable_name;

format specifier of float is "%f"

float b;
scanf("%f",&b);
```

## **Double type**

printf("%f",b);

To store a double-precision floating-point value

```
double variable_name;
format specifier of double is "%f"
  double ch;
  scanf("%lf",&ch);
  printf("%lf",ch);
```

# Void type

Represents the absence of the type

void

# Zaroorat e rishta

Zaroorat e rishta is not a problem now with shaadeepk

shaadee.pk

0

## **Escape Sequences**

It is a sequence of characters starting with a backslash, and it doesn't represent itself when used inside string literal.

## Alarm or Beep

It produces a beep sound

```
#include<stdio.h>
int main()
{
printf("\a");
return 0;
}
```

# **Backspace**

It adds a backspace

```
#include<stdio.h>
int main()
{
printf("\b");
return 0;
}
```

#### Form feed

```
#include<stdio.h>
int main()
{
printf("\f");
return 0;
}
```

#### Newline

Newline Character

```
#include<stdio.h>
int main()
{
printf("\n");
return 0;
}
```

# Carriage return

```
#include<stdio.h>
int main()
{
printf("\r");
return 0;
}
```

#### Tab

It gives a tab space

```
#include<stdio.h>
int main()
```

```
{
printf("\t");
return 0;
}
```

#### **Backslash**

It adds a backslash

```
#include<stdio.h>
int main()
{
printf("\\");
return 0;
}
```

# Single quote

It adds a single quotation mark

```
#include<stdio.h>
int main()
{
printf("\'");
return 0;
}
```

## **Question mark**

It adds a question mark

```
#include<stdio.h>
int main()
{
printf("\?");
return 0;
}
```

#### Octal No.

It represents the value of an octal number

```
#include<stdio.h>
int main()
{
printf("\nnn");
return 0;
}
```

## Hexadecimal No.

It represents the value of a hexadecimal number

```
#include<stdio.h>
int main()
{
printf("\xhh");
return 0;
}
```

### Null

The null character is usually used to terminate a string

```
#include<stdio.h>
int main()
{
printf("\0");
return 0;
```

## **Conditional Instructions**

Conditional statements are used to perform operations based on some condition.

#### If Statement

```
if (/* condition */)
{
/* code */
}
```

#### **If-else Statement**

```
if (/* condition */)
{
  /* code */
}
else{
  /* Code */
}
```

## if else-if Statement

```
if (condition) {
// Statements;
}
else if (condition){
// Statements;
}
else{
// Statements
}
```

#### nested if-else

```
if (/* condition */)
{
  if (/* condition */)
{
  /* code */
}
  else{
  /* Code */
}
  else{
  /* Code */
}
```

#### **Switch Case Statement**

It allows a variable to be tested for equality against a list of values (cases).

```
switch (expression)
{
  case constant-expression:
  statement1;
  statement2;
  break;
  case constant-expression:
  statement;
  break;
  ...
  default:
  statement;
}
```

# Zaroorat e rishta

Zaroorat e rishta is not a problem now with shaadeepk

shaadee.pk

#### 0

(i

## **Iterative Statements**

Iterative statements facilitate programmers to execute any block of code lines repeatedly and can be controlled as per conditions added by the programmer.

## while Loop

It allows execution of statement inside the block of the loop until the condition of loop succeeds.

```
while (/* condition */)
{
/* code */
}
```

## do-while loop

It is an exit controlled loop. It is very similar to the while loop with one difference, i.e., the body of the do-while loop is executed at least once even if the expression is false

```
do
{
/* code */
} while (/* condition */);
```

## for loop

It is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

```
for (int i = 0; i < count; i++)
{
/* code */
}</pre>
```

#### **Break Statement**

break keyword inside the loop is used to terminate the loop

```
break;
```

#### Continue Statement

continue keyword skips the rest of the current iteration of the loop and returns to the starting point of the loop

```
continue;
```

## **Functions & Recursion**

Functions are used to divide an extensive program into smaller pieces. It can be called multiple times to provide reusability and modularity to the C program.

#### **Function Definition**

```
return_type function_name(data_type parameter...){
//code to be executed
}
```

#### **Function Call**

```
Function_name(parameters...);
```

#### return\_type in functions

The function return statement return the specified value or data item to the caller. If we do not want to return any value simply place a void before function name while defining it.

```
return_type function_name()
{
return value;
}
```

#### Parameters in python function

Parameters are the values passed inside the parenthesis of the function while defining as well as while calling.

```
return_type function_name(data_type parameter...){    //defining the functions with parameters
//code to be executed
}
function_name(parameter...);    //calling the functions with parameters
```

# ways of calling a function

- 1. With return value and with parameters
- 2. without return value and with parameters
- 3. with return value and without parameters
- 4. without return value and without parameters

#### Recursion

Recursion is when a function calls a copy of itself to work on a minor problem. And the function that calls itself is known as the Recursive function.

```
void recurse()
{
....
recurse();
```

```
}
```

## **Pointers**

Pointer is a variable that contains the address of another variable,

## Declaration

```
datatype *var_name;
```

we can allocate the address of pointing variable to the pointer variable

```
#include<stdio.h>
int main()
{
  int *ptr,x;
  x=15;
  ptr=&x;
  printf("%d",ptr);
  return 0;
}
```

# dereferencing pointer variable

```
#include<stdio.h>
int main()
{
  int *ptr,x;
  x=12;
  printf("%d",*ptr);
  return 0;
}
```

# **Arrays**

An array is a collection of data items of the same type.

#### Declaration

```
data_type array_name[array_size];
#include<stdio.h>
int main()
{
int arr[10];
}
```

## Accessing element

```
data_type variable_name = array[index];
```

# Zaroorat e rishta

Zaroorat e rishta is not a problem now with shaadeepk

shaadee.pk

0

# **Strings**

A string is a 1-D character array terminated by a null character ('\0')

#### Declaration

```
char str_name[size];
```

# gets() function

It allows you to enter multi-word string

```
gets("string");
```

## puts() function

It is used to show string output

```
puts("string");
```

# String Functions strlen()

It is used to calculate the length of the string

```
strlen(string_name);
```

## strcpy() function

It is used to copy the content of second-string into the first string passed to it

```
strcpy(destination, source);
```

## strcat() function

It is used to concatenate two strings

```
strcat(first_string, second_string);
```

## strcmp() function

It is used to compare two strings

```
strcmp(first_string, second_string);
```

#### strlwr() function

It is used to covert characters of strings into lowercase

```
strlwr(string_name);
```

## strupr() function

It is used to covert characters of strings into uppercase

```
strupr(string_name);
```

### strrev() function

It is used to reverse the string

```
strrev(string_name);
```

#### **Structures**

The structure is a collection of variables of different types under a single name. Defining structure means creating a new data type.

#### Structure syntax

```
struct structureName
{
dataType member1;
dataType member2;
...
};
```

## typedef keyword

typedef function allows users to provide alternative names for the primitive and user-defined data types.

```
typedef struct structureName
{
dataType member1;
dataType member2;
...
}new_name;
```

# Zaroorat e rishta

Zaroorat e rishta is not a problem now with shaadeepk

shaadee.pk

0

# File Handling

A set of methods for handling File IO (read/write/append) in C language

#### FILE pointer

```
FILE *filePointer:
```

# Opening a file

It is used to open file in C.

```
filePointer = fopen(fileName.txt, w)
```

#### fscanf() function

It is used to read the content of file.

```
fscanf(FILE *stream, const char *format, ...)
```

## fprintf() function

It is used to write content into the file.

```
fprintf(FILE *fptr, const char *str, ...);
```

## fgetc() function

It reads a character from a file opened in read mode. It returns EOF on reaching the end of file.

```
fgetc(FILE *pointer);
```

## fputc() function

It writes a character to a file opened in write mode

```
fputc(char, FILE *pointer);
```

### Closing a file

It closes the file.

```
fclose(filePointer);
```

# **Dynamic Memory Allocation**

A set of functions for dynamic memory allocation from the heap. These methods are used to use the dynamic memory which makes our C programs more efficient

#### malloc() function

Stands for 'Memory allocation' and reserves a block of memory with the given amount of bytes.

```
ptr = (castType*) malloc(size);
```

#### calloc() function

Stands for 'Contiguous allocation' and reserves n blocks of memory with the given amount of bytes.

```
ptr = (castType*)calloc(n, size);
```

#### free function

It is used to free the allocated memory.

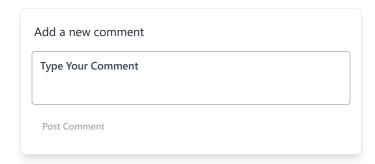
```
free(ptr);
```

#### realloc() function

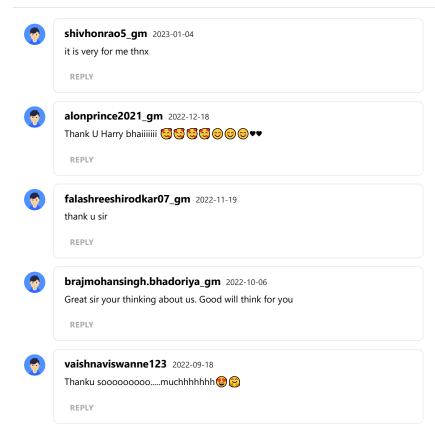
If the allocated memory is insufficient, then we can change the size of previously allocated memory using this function for efficiency purposes

```
ptr = realloc(ptr, x);
```

**Download this Cheatsheet** 



# Comments (9)





Copyright © 2022 CodeWithHarry.com



