C Cheat Sheet: &

Learn Now

Boiler plate

```
#include //header files
int  main()  //main function
{
      // Your code here
      return(0);  //returning value to int main()
}
```

Printf

It is used to show output on the screen

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

Scanf

It is used to take input from the user

```
scanf("format_specifier", &variables)
```

Example: -

```
int  a;
scanf("%d",&a);  // Store keyboard input in a variable with
address (address of a or &a)
printf("%d",&a);
```

Comments

A comment is a code that is not executed by the compiler, and the programmer uses it to annotate their code, providing explanations or reminders about the code's functionality, which aids in readability and future maintenance.

Single Line Comment

```
// This is a single line comment
```

Single Line Comment

```
/* This is a
multi-line
comment
*/
```

Data Types

The data type defines the kind of data that can be stored in a variable, such as integers, floating-point numbers, characters, or more complex structures. It dictates how the data is stored, interpreted, and manipulated within the program.

Character type

The character type, often represented as a single octet (one byte), is used to store individual characters in the C programming language.

```
char variable_name;
```

The format specifier for a character in C is "%c". To print a character, we use this specifier within the printf function, following the syntax like this:

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

Integer type

To store non-decimal numeric values, an integer type is used

```
int variable_name;
```

The format specifier of an 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;
```

The format specifier of a float is "%f"

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

Double type

To store a double-precision floating-point value we use double.

```
double variable_name;
```

The format specifier of double is "%f"

```
double ch;
scanf("%lf",&ch);
printf("%lf",ch);
```

Void type

The void type in C represents the absence of a type. It's often used in function declarations to specify that the function does not return any value. For example:

```
void myFunction() {
// Function code here
}
```

In this context, the void keyword indicates that myFunction does not return a value. It can also be used for function parameters to indicate that a function takes no arguments

Escape Sequences

Escape sequences in C are combinations of characters that begin with a backslash (\) and are used to represent characters that cannot be typed directly. These sequences are interpreted in a special way when used inside string literals or character constants.
For example, the escape sequence \n represents a newline character, and \t represents a tab character. Here are some escape sequence characters used in C language.

Alarm or Beep

\a produces a beep sound

```
#include
int  main()
{
      printf("\a");  // It produces a beep sound
      return  0;
}
```

Backspace

\b adds a backspace

```
#include
int  main()
{
      printf("Hello\bWorld");  // It prints "HellWorld"
      return  0;
}
```

Form Feed

```
#include
int  main()
{
      printf("Page break here\fContinue text");  // It may create a
page break, but it's not supported everywhere
      return  0;
}
```

Newline

Newline Character

```
#include
int  main()
{
      printf("Line one\nLine two");  // Prints two lines
      return  0;
}
```

Carriage return

The carriage return, represented by the escape sequence \r in the C programming language, is a control character that resets the cursor position to the beginning of the current line. It doesn't erase any characters but simply moves the cursor to the start of the line. The string "Hello" is printed first, then the carriage return moves the cursor back to the beginning of the line, and "World" is printed, overwriting "Hello."

```
#include
int  main()
{
      printf("Hello\rWorld");  // Outputs "World" but behavior might vary depending on the OS
      return  0;
}
```

Tab

It gives a tab space

```
#include
int  main()
{
      printf("Tabbed\ttext");  // Adds a tab space
      return  0;
}
```

Backslash

It adds a backslash

```
#include
int  main()
{
      printf("\\"); // Prints a backslash
      return  0;
}
```

Single quote

It adds a single quotation mark

```
#include
int  main()
{
      printf("\'"); // Prints a single quotation mark
      return  0;
}
```

Question mark

It adds a question mark

```
#include
int  main()
{
      printf("\?"); // Prints a question mark
      return  0;
}
```

Octal No.

It represents the value of an octal number

```
#include
int  main()
{
      printf("\101");  // Prints 'A', which is 101 in octal
      return  0;
}
```

Hexadecimal No.

It represents the value of a hexadecimal number

```
#include
int  main()
{
      printf("\x41");  // Prints 'A', which is 41 in hexadecimal
      return  0;
}
```

NULL

The null character is usually used to terminate a string

```
#include
int  main()
{
      printf("\0");
      char  str[]  =  "Hello\0World";  // The null character is used
to terminate a string
      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

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

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 the execution of statements inside the block of the loop until the condition of the 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 */
}
```

Break Statement

break keyword inside the loop is used to terminate the loop

```
#include

int main() {
    for (int i = 0; i < 10; i++) {
        if (i == 5) {
            printf("Loop is breaking at i = 5\n");
            break;  // Exit the loop when i is 5
        }
        printf("i = %d\n", i);
    }

    return 0;
}
```

Continue Statement

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

```
#include

int main() {
    for (int i = 1; i <= 10; i++) {
        if (i % 2 == 0) {
            continue; // Skip the rest of the loop body if i is
even
        }
        printf("%d ", i); // Print the odd numbers
    }
    return 0;
}

// Output is 1 3 5 7 9
```

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 Function

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

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

Recursion

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

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

Pointers

A pointer is a variable that contains the address of another variable,

Declaration

```
datatype *var_name;
```

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

```c
#include

int main() {
    int *ptr, x;
    x = 15;
    ptr = &x;

    // This will print the address of x, not the value 15
    printf("%p", ptr);

    return 0;
}
```

Dereferencing pointer variable

```c
#include

int main() {
    int *ptr, x;
    x = 12;
    ptr = &x; // Assign the address of x to ptr
    printf("%d", *ptr); // Dereference ptr to print the value of x

    return 0;
}
```

Arrays

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

Declaration

```c
data_type array_name[array_size];
```

```c
#include
int main()
{
int arr[10];
}
```

Accessing element

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

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 a multi-word string.

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

puts() function

It is used to show string output

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

fgets() function

The gets() function is considered unsafe, and it is better to use fgets() instead.

```
#include

int main() {
    char str[50];
    printf("Enter a string: ");
```

```
    fgets(str, sizeof(str), stdin);
    printf("You entered: %s", str);
    return 0;
}
```

String Functions

strlen() function

It is used to calculate the length of the string

```
strlen(string_name);
```

strcpy() function

It is used to 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 convert characters of strings into lowercase

```
strlwr(string_name);
```

strupr() function

It is used to convert 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;
```

File Handling

FILE Pointer

```
FILE *filePointer;
```

Opening a file

It is used to open a file in C.

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

fscanf() function

It is used to read the content of a 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 the 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);
```

CHEAT SHEET

Developed by Pratham Choudhary