Enhanced C Programming Guide (W3Schools)

Based on W3Schools

August 13, 2025

Contents

Chapter 1

Get Started

C is a procedural programming language used for system programming. It provides low-level memory access, making it powerful but prone to errors if misused.

Features of C

- Portable: Programs can run on any machine with little or no change.
- Fast execution due to low-level access.
- Widely used in embedded systems, OS kernels, and hardware drivers.

Structure of a C Program

```c
#include <stdio.h> // Preprocessor directive

int main() {
    printf("Hello, World!\n"); // Function call
    return 0; // Exit status
}
```

Chapter 2

Syntax and Compilation

A C program goes through preprocessing, compilation, assembly, and linking.

Compilation Pipeline

1. Preprocessing (.i)
2. Compilation to Assembly (.s)
3. Assembly to Object Code (.o)
4. Linking to Executable

Compilation command:

```
gcc hello.c -o hello
./hello
```

Chapter 3

Input and Output

printf() Function

Used for formatted output.

```c
int x = 10;
printf("Value: %d\n", x);
```

Format specifiers:

- %d - int
- %f - float
- %c - char
- %s - string

scanf() Function

Used for input from the user.

```c
int age;
scanf("%d", &age); // Note the & operator
```

Chapter 4

Variables and Data Types

Variables are declared with a data type. Types include:

- `int` – integers (4 bytes)
- `float`, `double` – real numbers
- `char` – single characters

Type Qualifiers

- `unsigned int`
- `long int`
- `short int`

Example

```c
int age = 25;
float pi = 3.14;
char grade = 'A';
```

Chapter 5

Constants

Use `const` to declare constants.

```
const float PI = 3.14159;
```

Also use `#define`:

```
#define MAX 100
```

Chapter 6

Operators

Arithmetic: +, -, *, /, %
Assignment: =, +=, -=
Relational: ¿, ¡, ==, !=
Logical: &&, ——, !
Bitwise: &, —, ˆ <<, >>

```
int a = 5, b = 2;
int c = a + b;
```

Chapter 7

Control Structures

If / Else

```c
if (x > y) {
  // code
} else {
  // code
}
```

Switch

```c
switch (day) {
  case 1:
    printf("Mon");
    break;
  default:
    printf("Invalid");
}
```

Chapter 8

Loops

While Loop

```c
int i = 0;
while (i < 5) {
    printf("%d\n", i);
    i++;
}
```

For Loop

```c
for (int i = 0; i < 5; i++) {
    printf("%d\n", i);
}
```

Do...While

```c
int i = 0;
do {
    printf("%d\n", i);
    i++;
} while (i < 5);
```

Break and Continue

```c
for (int i = 0; i < 5; i++) {
  if (i == 3) break;
  if (i == 1) continue;
  printf("%d\n", i);
}
```

Chapter 9

Arrays

Definition: Arrays are fixed-size sequential collections of same data type.

```c
int nums[5] = {1, 2, 3, 4, 5};
printf("%d", nums[0]);
```

Traversal:

```c
for (int i = 0; i < 5; i++)
    printf("%d ", nums[i]);
```

Chapter 10

Strings

Stored as character arrays ending with ‘\0‘.

```c
char str1[] = "Hello";
char str2[6] = {'H','e','l','l','o','\0'};
```

String Functions:

- `strlen(str)`
- `strcpy(dest, src)`
- `strcat()`
- `strcmp()`

Chapter 11

User Input

Use `scanf` and `gets` (unsafe) or `fgets`.

```c
char name[30];
printf("Enter name: ");
scanf("%s", name);
```

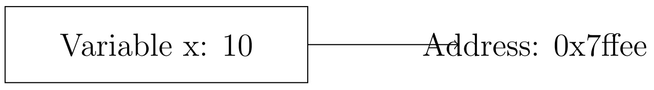
Chapter 12

Memory Address

gives memory address of a variable.

```c
int x = 10;
printf("Address: %p", &x);
```

Memory Diagram:



Chapter 13

Pointers

Pointers store memory addresses.

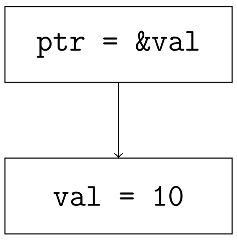
```c
int val = 10;
int *ptr = &val;

printf("%d\n", val);     // 10
printf("%p\n", ptr);     // address
printf("%d\n", *ptr);    // dereference
```

Pointer Notes

- ‘*‘ is used for dereferencing.
- “ is used to get the address.
- Pointers must be initialized before dereferencing.

Diagram


ptr = &val
val = 10