C

Pointer

Patrick Stiller

November 4, 2014

Introduction

To unterstand pointers you need some knowledge about the structure of storage.

Simple illustration

The memory is divided into large number of small parts called cells. Every cell has a adress. With the adress you have access to the value of the cell.The compiler connects the variables with the cells. The value of the cell is the value of the variable.

variable	value of variable	address(hex)
		FFFF
		FFFE
		...
int x	50	456F
		456E

Adresses

If you want the adresses of variables you need the address-of-operator(&).

```
#include <stdio.h>

int main(int argc, char *argv[]){
      //small programm printing the address of x
      int x = 50;
      printf("address of x : %x",&x); // address in hex
      return 0;
}
```

- Pointers are the data types of addresses
- The value of a pointer is an address.
- To define a pointer you need the star-operator(*)
- Pointers have an one address.

Syntax

```
dataType* pointerToDataType ;
```

examples

```
  //define some pointer with different data types
  int* pointerToInt;
  double* pointerToDouble;
  char** pointerToPointerToChar;
  struct Tag* pointerToStructTag;

```

Value assignment

The value of a pointer is an address.

```
int x = 50;
int* a;

pointerToInt = &x ;
//pointerToInt exhibits to x
```

Pointer in memory

Pointers have an one address.

variable	value of variable	address(hex)
		FFFF
		FFFE
		...
int x	50	456F
int*a	456F	456E

NULL Pointer

- NULL is an invalid address
- A pointer without aim

```
int* pointerToInt = NULL;
int* pointerToDouble = NULL;
char** pointerToPointerToChar = NULL;
```

Access to NULL is not possible. Do produce safe code!

```
if(pointerToInt &&(pointerToDouble != NULL){
// neither pointerToInt nor pointerToDouble is NULL
}
```

Dereferencing

- Access to the aim of the pointer
- The dereferencing of NULL pointers is not allowed
- To get access you need the star-operator(*)before the pointer variable

```
int anInt = 12;
int anAnotherInt;
int anPointer = &anInt;

*anPointer = *anPointer + 12;
*anPointer = *anPointer * 2;
anAnotherInt = *anPointer ;
printf("Value of anInt %d",anInt); //prints 48
printf("Value of anAnotherInt %d",anInt); //prints 48
```

Pointers as parameters

```
1  void double_arguments(int* parameter){
2    *parameter *= 2;
3  }
4
5  int anInt = 23;
6
7  int main(int argc, char *argv[]){
8    double_argument(&anInt);
9    printf("%d \n",anInt); // 46
10   return 0;
11 }
```