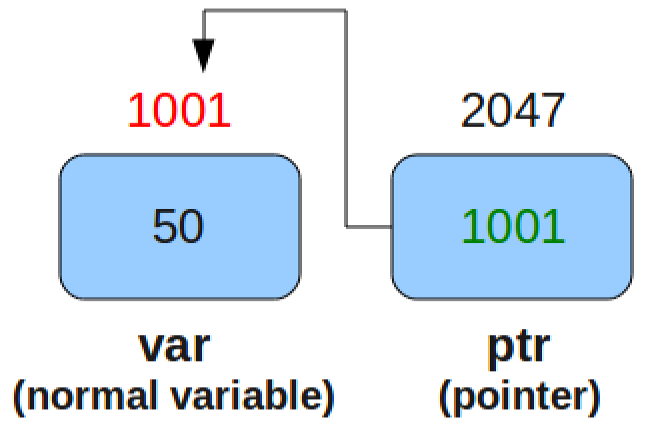
Pointer in C

Acknowledgement: programize.com



Warning: It is just a lecture purpose ppt file. Do not use it for the preparation of exam.

Md. Hasibur Rahman, Lecturer, Dept. of CSE, BUBT.

Address in C

scanf("%d", &var);

Address of the variable ***var***

```
#include <stdio.h>
int main()
{
         int var = 5;
        printf("Value: %d\n", var);
        printf("Address: %u", &var); //Notice,
        the ampersand(&) before var.
        return 0;
}
```

Output

```
Value: 5
Address: 2686778
```

Pointer Variables

Creating Pointer Variable:

```
data_type*pointer_variable_name;
int* p;
```

Reference operator **(&):** gives **address of a variable**

Dereference operator **(*):** gives **the value of a variable**

Example: How Pointer Works?

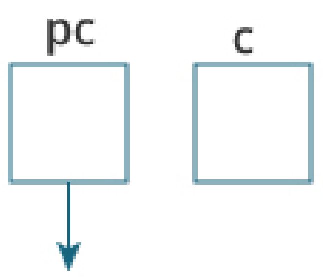
```
#include <stdio.h>
int main()
{
   int* pc, c;
   c = 22;
   printf("Address of c: %u\n", &c);
   printf("Value of c: %d\n\n", c);
   pc = &c;
   printf("Address of pointer pc: %u\n",
pc);
   printf("Content of pointer pc:
%d\n\n", *pc);
   c = 11;
   printf("Address of pointer pc: %u\n",
pc);
```

```
   *pc = 2;
   printf("Address of c: %u\n", &c);
   printf("Value of c: %d\n\n", c);
   return 0;
}
```

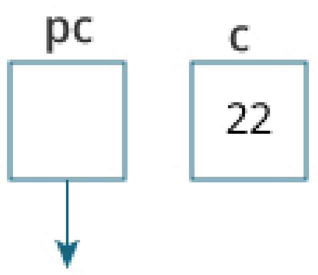
```
Address of c: 2686784

Value of c: 22

Address of pointer pc: 2686784

Content of pointer pc: 22

Address of pointer pc: 2686784

Content of pointer pc: 11

Address of c: 2686784

Value of c: 2
```

Explanation of the Program

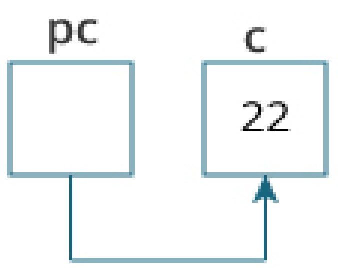
1. int* pc, c;



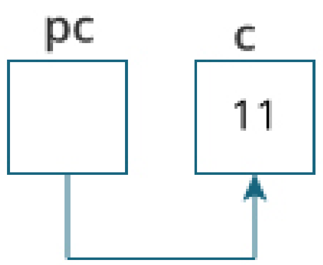
2. c = 22;



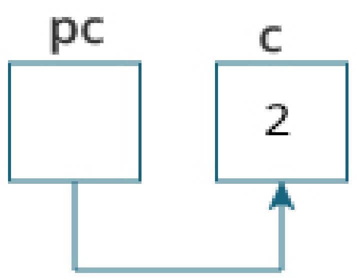
3. pc = &c;



4. c = 11;



5. *pc = 2;



Common mistakes when working with pointers

```
int c, *pc;

// Wrong! pc is address whereas,
// c is not an address.
pc = c;

// Wrong! *pc is the value
pointed by address whereas,
// &c is an address.
*pc = &c;
```

```
// Correct! pc is an address and,
// &c is also an address.
pc = &c;

// Correct! *pc is the value pointed by addres
and,
// c is also a value (not address).
*pc = c;
```

Case Study

OPERATOR	TYPE	ASSOCIAVITY
( ) [ ] . ->		left-to-right
++ -- + - ! ~ (type) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

```
// PROGRAM 1
#include <stdio.h>
int main(void)
{
    int a = 10;
    int *p = &a;
    printf(“%d”,*p);
    ++*p;
    printf(“%d”,*p);
    return 0;
}
```

```
// PROGRAM 2
#include <stdio.h>
int main(void)
{
    int a = 10;
    int *p = &a;
    *p++;
    printf(“%d”,*p);
    return 0;
}
```

P=10485316

C Pointers and Arrays

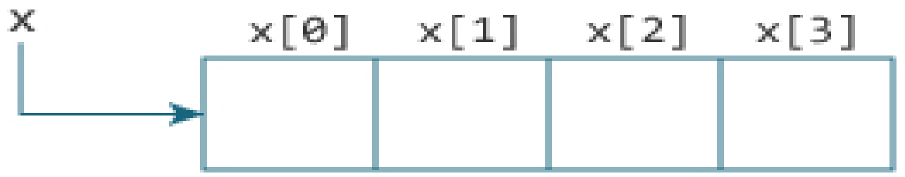
```
#include <stdio.h>
int main()
{
   int x[4];
   int i;
   for(i = 0; i < 4; ++i)
   {
      printf("&x[%d] = %u\n", i,
&x[i]);
   }
   printf("Address of array x:
%u", x);
   return 0;
}
```

<u>Output:</u>

```
&x[0] = 1450734448
&x[1] = 1450734452
&x[2] = 1450734456
&x[3] = 1450734460
Address of array x:
1450734448
```

Relation between Arrays and Pointers

int x[4];



x[0] is equivalent to *x

- **&x[1]** is equivalent to **x+1** and **x[1]** is equivalent to ***(x+1).**
- **&x[2]** is equivalent to **x+2** and **x[2]** is equivalent to ***(x+2).**
- ...
- Basically, **&x[i]** is equivalent to **x+i** and **x[i]** is equivalent to ***(x+i).**

Example 1: Arrays and Pointers

```
#include <stdio.h>
int main()
{
   int data[5], i;
   printf("Enter elements: ");
   for(i = 0; i < 5; ++i)
     scanf("%d", data + i);

   printf("You entered: \n");
   for(i = 0; i < 5; ++i)
      printf("%d\n", *(data + i));
   return 0;
}
```

Output:

```
Enter elements:
1
2
3
5
4
You entered:
1
2
3
5
4
```

Example 2: Arrays and Pointers

```
#include <stdio.h>
int main()
{
  int x[5] = {1, 2, 3, 4, 5};
  int* ptr;

  ptr = &x[2];

  printf("*ptr = %d \n", *ptr);
  printf("*ptr+1 = %d \n", *ptr+1);
  printf("*ptr-1 = %d", *ptr-1);
  return 0;
}
```

Output:

```
*ptr = 3

*ptr+1 = 4

*ptr-1 = 2
```

Example 3: Pointers and Arrays

```
#include <stdio.h>
int main()
{
  int i, x[6], sum = 0;
  printf("Enter 6 numbers: ");
  for(i = 0; i < 6; ++i)
  {
      scanf("%d", x+i);

      sum += *(x+i);
  }
  printf("Sum = %d", sum);
  return 0;
}
```

Output:

```
Enter 6 numbers:
2
 3
 4
 4
 12
 4
Sum = 29
```

Function: Pass By Value

```
#include <stdio.h>

void swapByValue(int, int); /*

Prototype */

int main() /* Main function */

{

  int n1 = 10, n2 = 20;

  /* actual arguments will be as it is */

  swapByValue(n1, n2);

  printf("n1: %d, n2: %d\n", n1, n2);

}
```

```
void swapByValue(int a, int b)
{
  int t;
  t = a; a = b; b = t;
}
```

```
OUTPUT
======
n1: 10, n2: 20
```

Function: Pass By Reference

```
#include <stdio.h>

void swapByReference(int*, int*);

/*Prototype */

int main() /* Main function */

{

  int n1 = 10, n2 = 20;

  /* actual arguments will be altered */

  swapByReference(&n1, &n2);

  printf("n1: %d, n2: %d\n", n1, n2);

}
```

```
void swapByReference(int *a, int
*b)
{
  int t;
  t = *a; *a = *b; *b = t;
}
```

```
OUTPUT
======
n1: 20, n2: 10
```

C Dynamic Memory Allocation

- Declare the size of an array before use it
- Sometimes, the size of array you declared may be insufficient
- Allocate memory manually during run-time
- malloc(), calloc(), realloc() and free()

C malloc()

- "malloc" stands for memory allocation.
- Syntax of malloc()

  ptr = (cast-type*) malloc(byte-size)
- Example

  ptr = (int*) malloc(100 * sizeof(int));

C Dynamic Memory Allocation

C calloc()

- “calloc" stands for *contigious* memory allocation.
- Syntax of calloc()

  ptr = (cast-type*) calloc(n, element-size)
- Example

  ptr = (int*) calloc(20, sizeof(float));

C free()

- explicitly use free() to release the space.
- Syntax: **free(ptr)**