Pointers

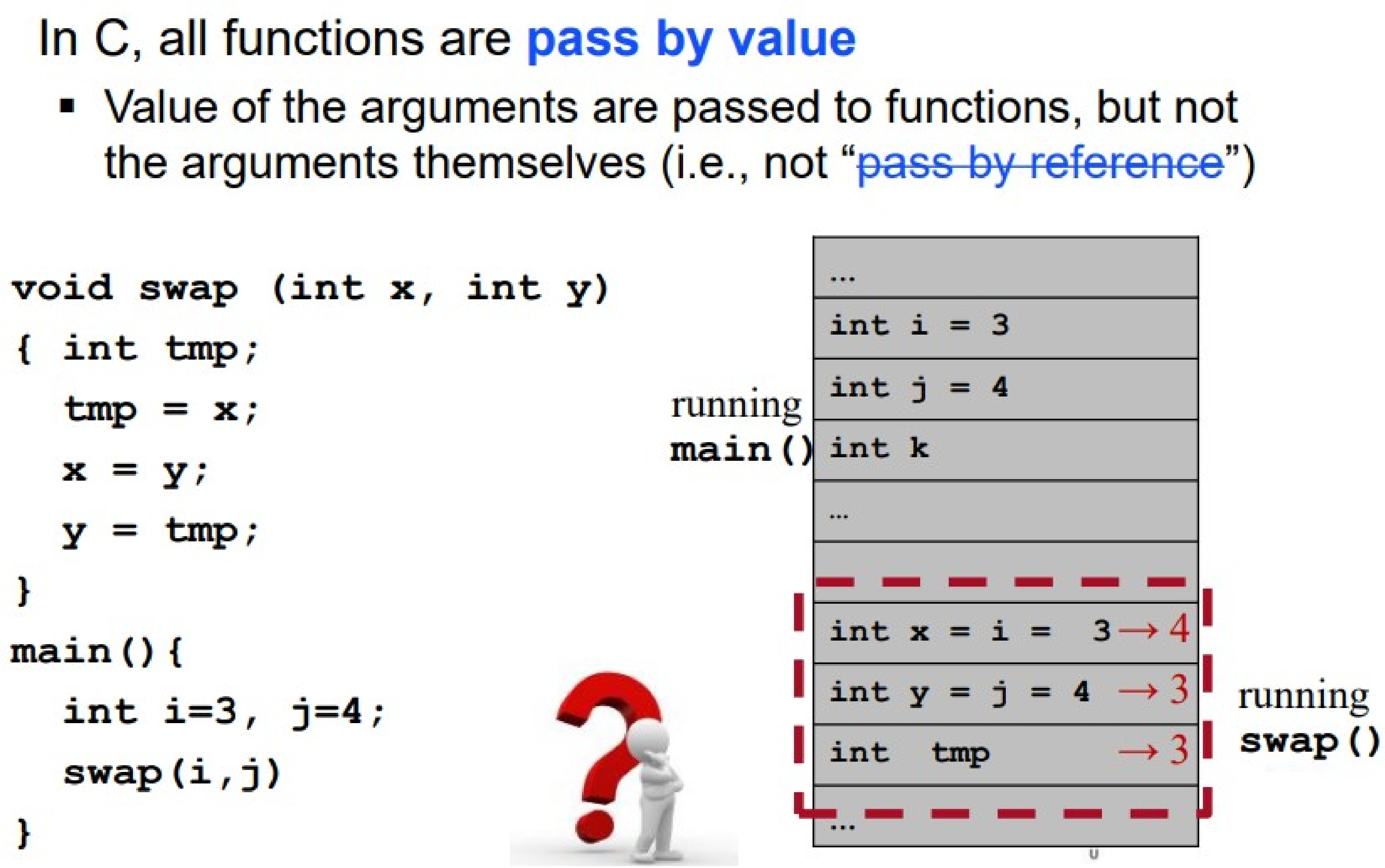
EECS 2031

Song Wang
wangsong@eecs.yorku.ca
eecs.yorku.ca/~wangsong/

Acknowledgement

- Some of the covered materials are based on previous EECS2031 offerings:
  - Uyen Trang (UT) Nguyen, Pooja Vashisth, Hui Wang, Manos Papagelis

Motivations: Pass-by-Value



In C, all functions are **pass by value**

- Value of the arguments are passed to functions, but not the arguments themselves (i.e., not "~~pass by reference~~")

```
void swap (int x, int y)
{ int tmp;
  tmp = x;
  x = y;
  y = tmp;
}
main(){
  int i=3, j=4;
  swap(i,j)
}
```

```
char fromStr [] = "Hello!";
char toStr [20];

strcpy(toStr, fromStr);     // toStr modified

fgets(toStr, 10, stdin);    // toStr modified
```

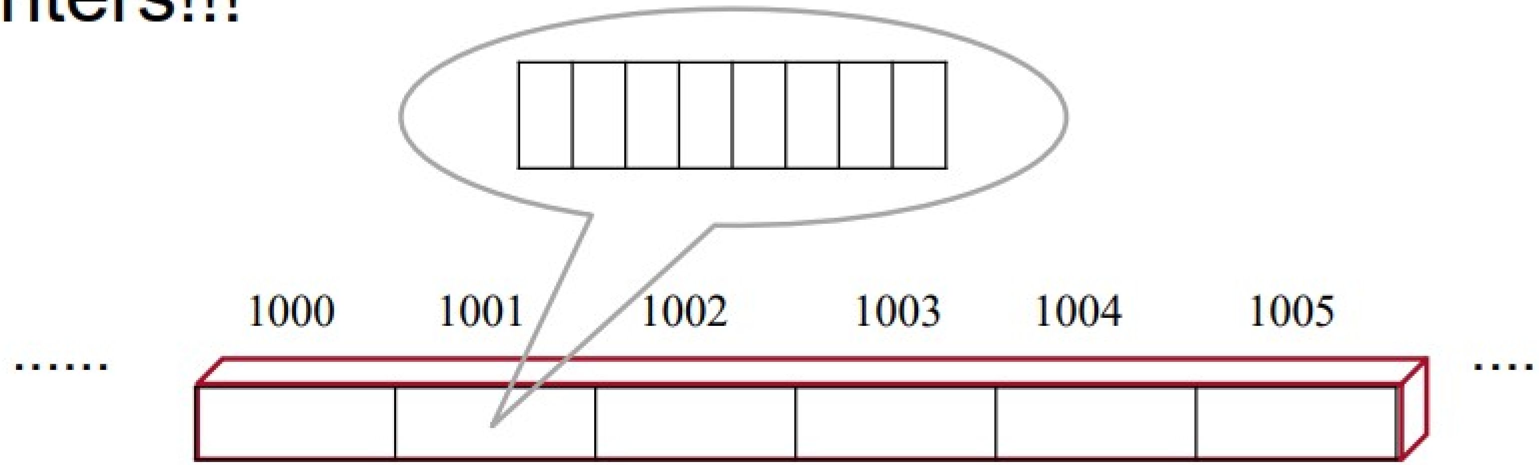
- Given an array as an argument, a function can modify the contents of the array -- Arrays are passed *as if* “call-by-reference”

- But isn’t C “call-by-value”? -- pass single numerical value
  - How to pass strings to **`strcpy()`**?
  - How does **`strcpy(),scanf(),fgets()`** modify argument?

- Also **`scanf ("%d %s", &a,  arr); // a arr modified`**
  - Why **`&a`**, why not **`&arr`**

- Why **`sizeof`** does not work in function call
  - return 8 or 4 always

Pointers!!!



- computers memory
  - Thousands of sequential storage location byte (8 bits)
  - Each byte has a unique address
  - Range 0 ~ max

Pointers

A *pointer* is a reference to another variable (memory location) in a program

- Used to change variables inside a function (reference parameters)
- Used to remember a particular member of a group (such as an array)
- Used in dynamic (on-the-fly) memory allocation (especially of arrays)
- Used in building complex data structures (linked lists, stacks, queues, trees, etc.)

Pointer Variable Definition

Basic syntax: _Type *Name_

Examples:

```
int *P;  /* P is var that can point to an int var */
float *Q;        /* Q is a float pointer */
char *R;         /* R is a char pointer */
```

Complex example:

```
int *AP[5];/* AP is an array of 5 pointers to ints */
```

Address (&) Operator

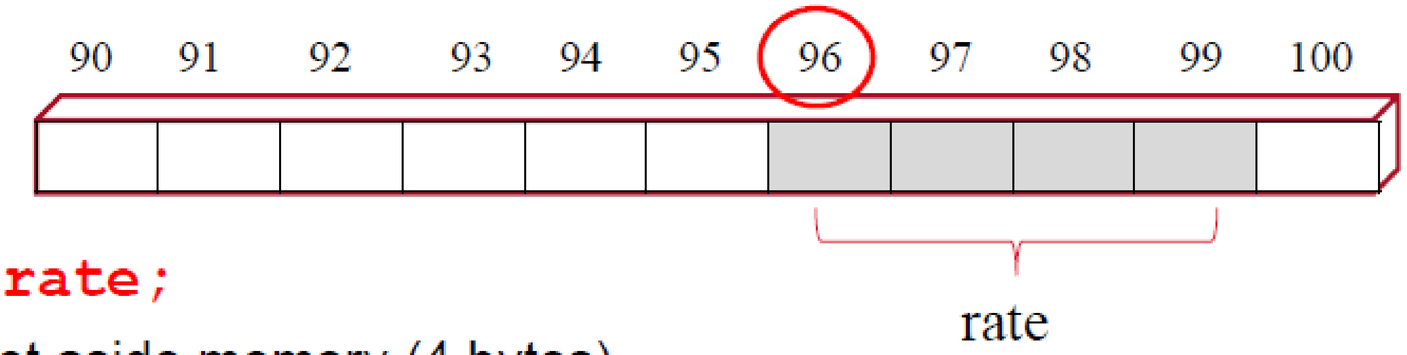
The address (&) operator can be used in front of any variable object in C -- the result of the operation is the location in memory of the variable

Syntax: ***&VariableReference***

Examples:

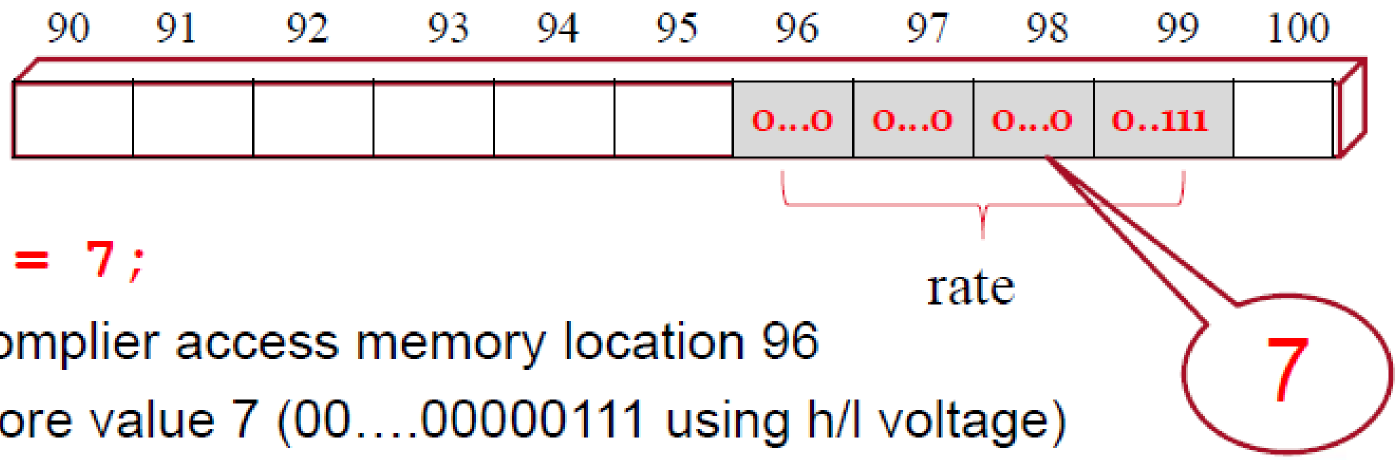
```
int V;
int *P;
int A[5];

&V - memory location of integer variable V
&(A[2]) - memory location of array element 2 in array A
&P - memory location of pointer variable P
```

Memory allocation for variable



```
int rate;
```

- set aside memory (4 bytes)
- associates 96 (starting address) with **rate**;



```
rate = 7;
```

- Complier access memory location 96
- Store value 7 (00….00000111 using h/l voltage)
- Hidden from you

C allows us to access and store the addresses of variables

Not in Java

&x

- address of a variable, array element. (No expression)

```
&x    &rate
&arr[0]; // later
scanf("%d %d", &a, &b);
```

---

type * p ;

- **p** is a pointer variable capable of storing the address of a int variable -- pointing to variable of type **type**

```
int * p, *q;
double * pd;
int j, a[10],  * p2, *q2;

p = ?
int *r = ?
```

C allows us to access and store the addresses of variables

Not in Java

```
&x
```

- address of a variable, array element. (No expression)

```
&x    &rate
&arr[0]; // later
scanf("%d %d", &a, &b);
```

```
type * p ;
```

- **p** is a pointer variable capable of storing the address of a int variable -- pointing to variable of type **type**

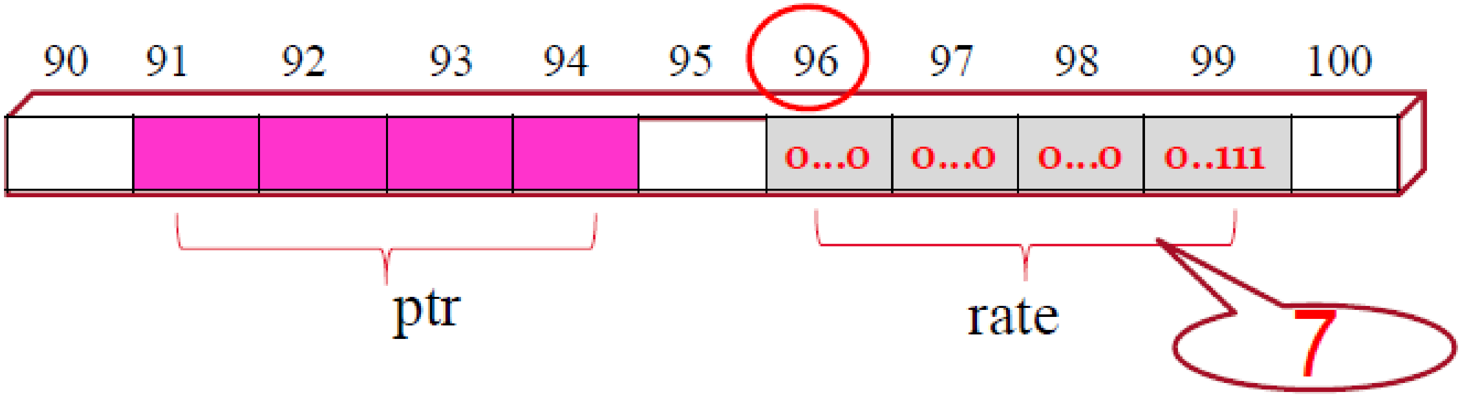
```
int * p, *q;
double * pd;
int j, a[10],  * p2, *q2;
```

```
p = &x;
int *r = &rate;
```

Declare and initialize pointer

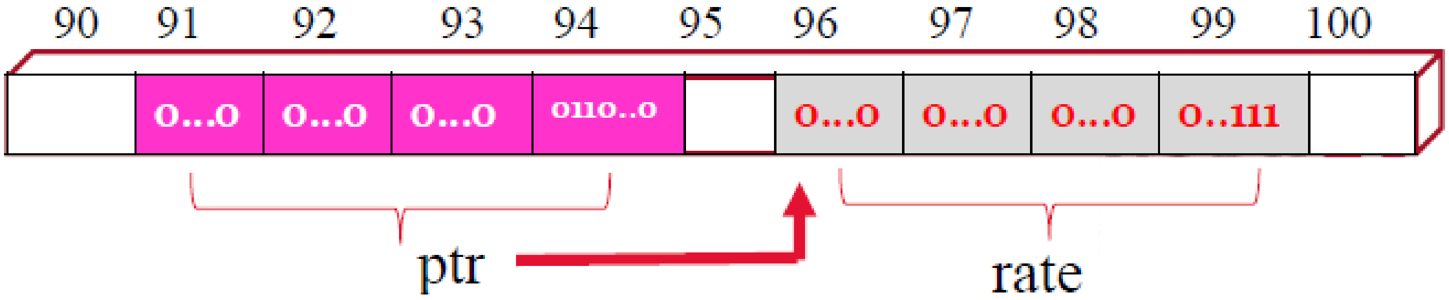
```
int *ptr;   /* declare a pointer to int  */
```

- Create a variable holding the address of other variable

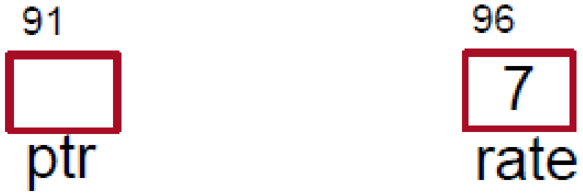


```
ptr = &rate  /*assigning address of rate*/
```

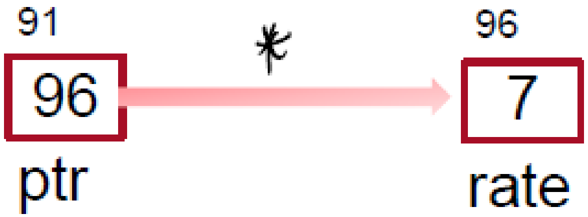
- Store address/pointer of `rate` in `ptr` (i.e., `ptr’`s value is the address)
- `ptr` now ‘points to’ `rate`



```
int *ptr;     /* I'm a pointer to an int */
```

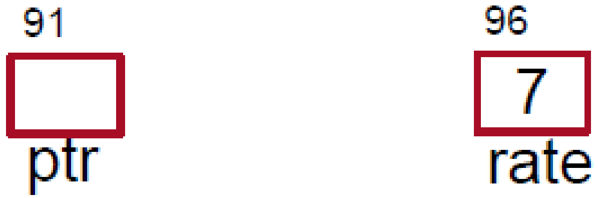


```
ptr = &rate; /*I got the address of rate */
```

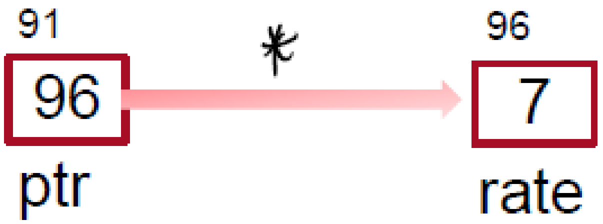


```
int *ptr;      /* I’m a pointer to an int */
```

mnemonic:
“expression *ptr
is an int”



```
ptr = &rate; /*I got the address of rate */
```

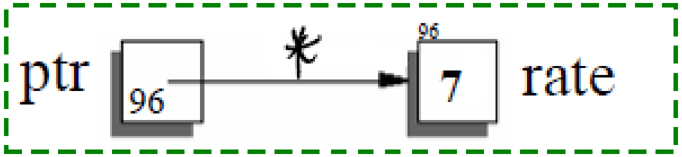


```
*ptr;      /* dereferencing. Indirect access.
             Get contents of the pointee */
```

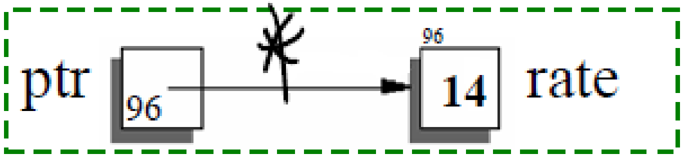
ptr	&rate	address of rate
*ptr	rate	content (value) of rate

```
printf("%d", rate);    // 7  "direct access"
printf("%d", *ptr);    // 7  "indirect access"
```

```
int main()
{
  int rate = 7;
  int *ptr = &rate;
```



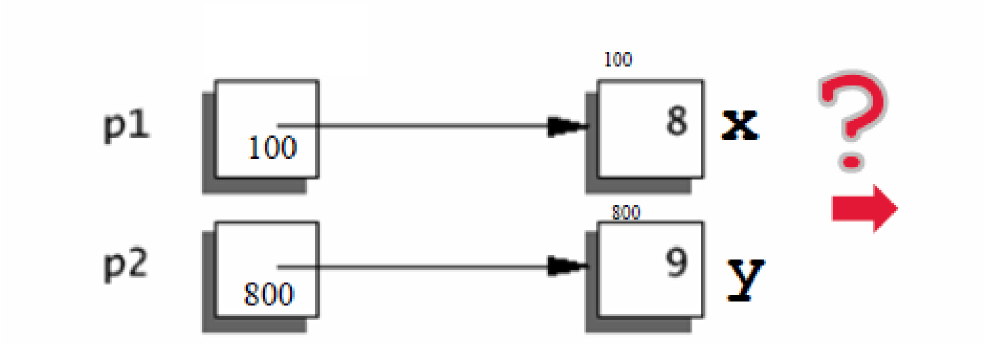
```
  printf("%d\n", rate); /* 7 */
  printf("%d\n", *ptr); /* 7 */

  int i = *ptr; // i=rate
```



```
  *ptr = 14; // rate = 14

  printf("%d %d\n", rate, *ptr); /* 14 14 */

  printf("%p %p\n", &rate, ptr); /*  96 96 */
}
```

Some examples of Pointer basics

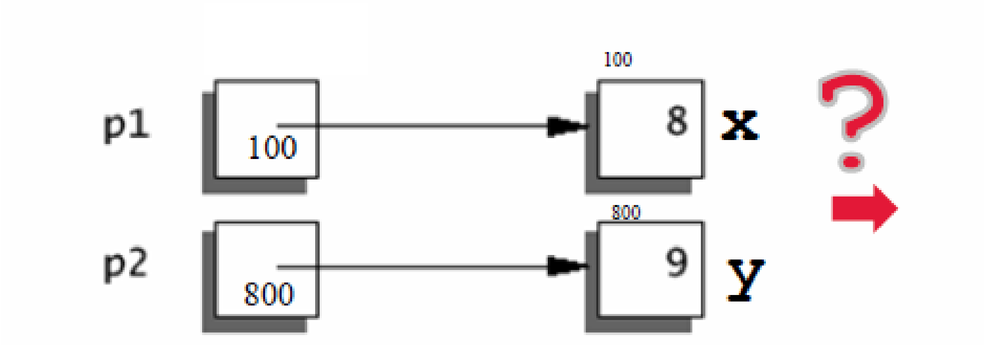
```
int *p1, *p2;  int x = 8, y = 9;
p1 = &x;  p2 = &y;
*p1 = *p2;    // x = y
```



Assume x is at address 100, y is at address 800

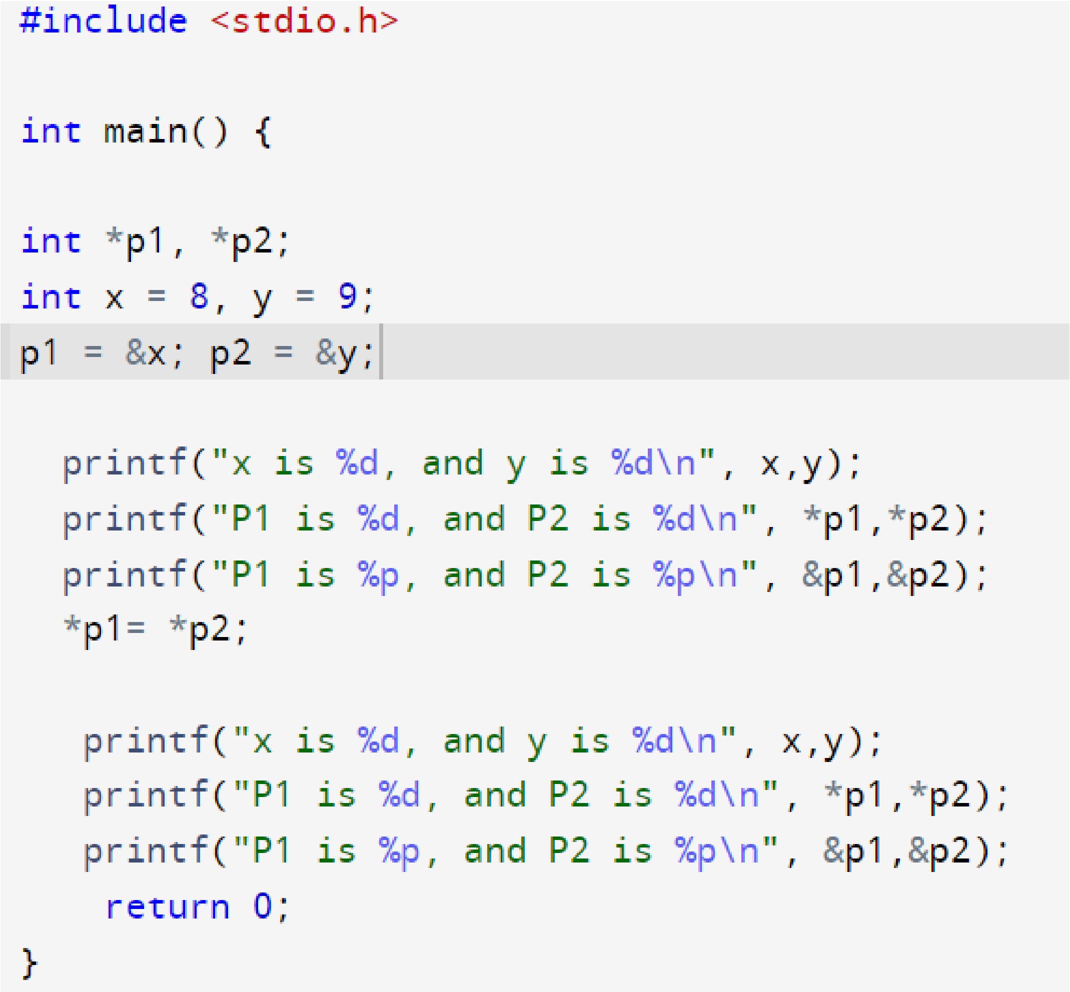
Some examples of Pointer basics

```
int *p1, *p2;  int x = 8, y = 9;
p1 = &x;  p2 = &y;
*p1 = *p2;    // x = y
```

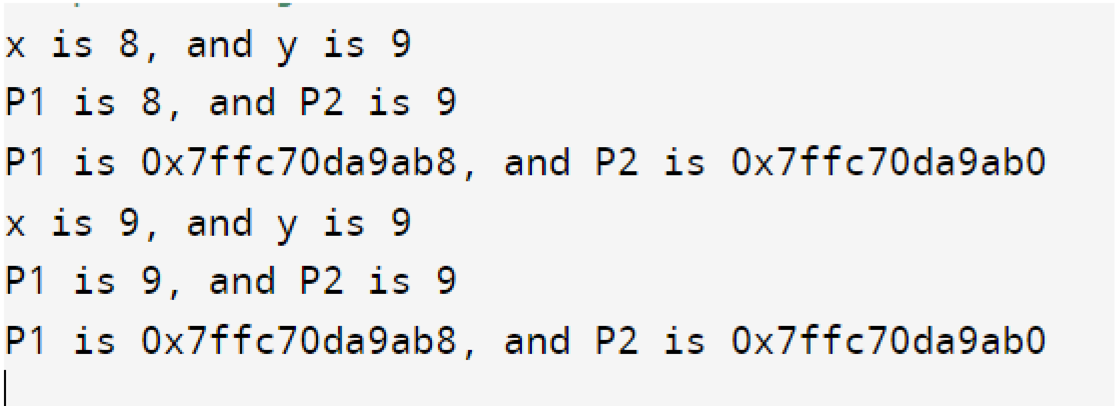


Assume x is at address 100, y is at address 800

```
// copy value of p2’s pointee(y) into pointeeof p1 (x)
```



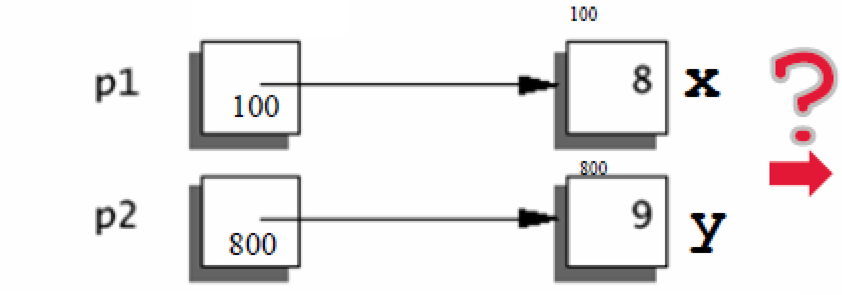
```c
#include <stdio.h>

int main() {

int *p1, *p2;
int x = 8, y = 9;
p1 = &x; p2 = &y;

  printf("x is %d, and y is %d\n", x,y);
  printf("P1 is %d, and P2 is %d\n", *p1,*p2);
  printf("P1 is %p, and P2 is %p\n", &p1,&p2);
  *p1= *p2;

   printf("x is %d, and y is %d\n", x,y);
   printf("P1 is %d, and P2 is %d\n", *p1,*p2);
   printf("P1 is %p, and P2 is %p\n", &p1,&p2);
    return 0;
}
```



```
x is 8, and y is 9
P1 is 8, and P2 is 9
P1 is 0x7ffc70da9ab8, and P2 is 0x7ffc70da9ab0
x is 9, and y is 9
P1 is 9, and P2 is 9
P1 is 0x7ffc70da9ab8, and P2 is 0x7ffc70da9ab0
```

Some examples of Pointer basics

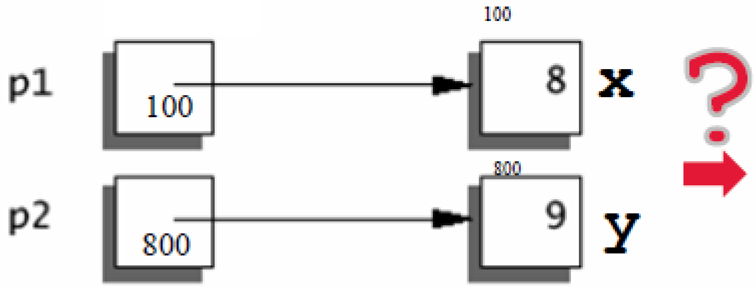
```
int *p1, *p2;  int x = 8, y = 9;
p1 = &x;  p2 = &y;
p1 = p2;    /*copy the content of p2 (address of y) into p1
             now p1 also points to y  */
```



Assume x is at address 100, y is at address 800

Some examples of Pointer basics

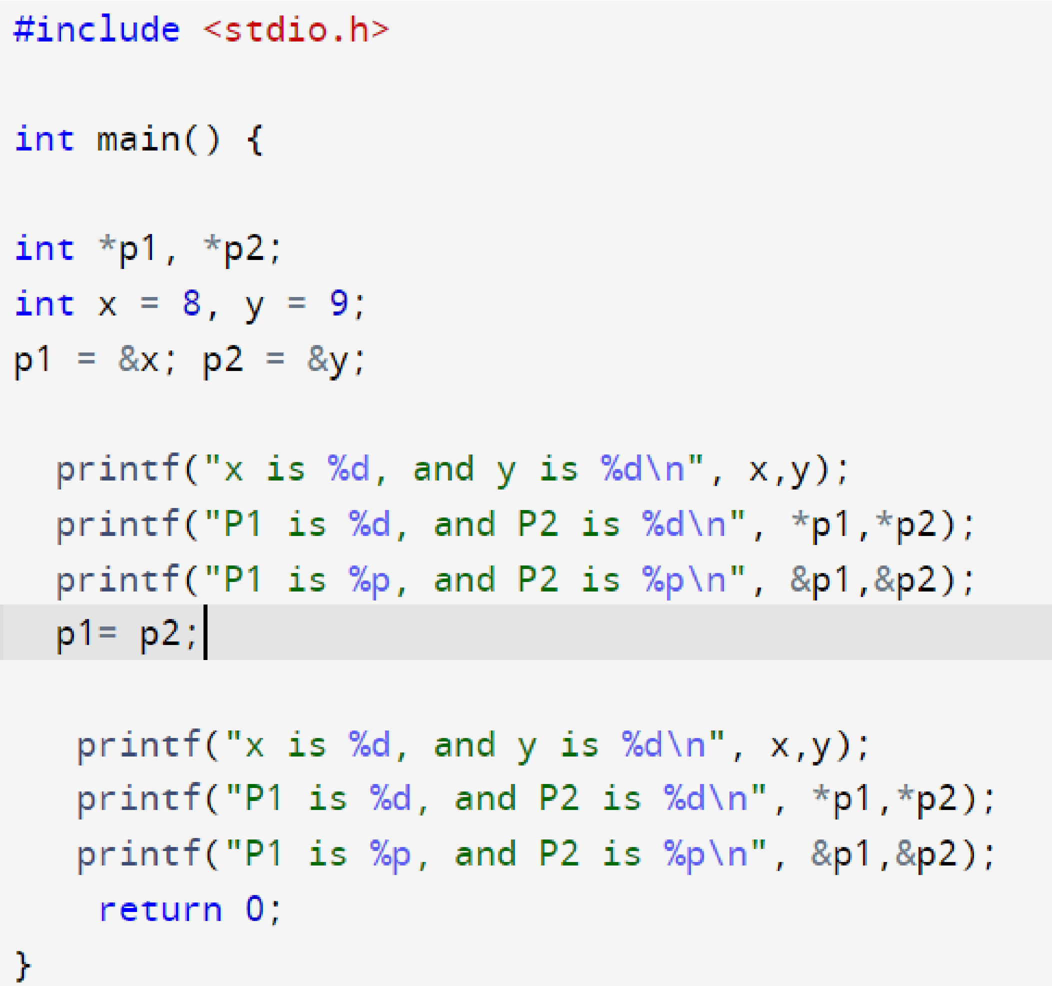
```
int *p1, *p2;  int x = 8, y = 9;
p1 = &x;  p2 = &y;
p1 = p2;    /*copy the content of p2 (address of y) into p1
              now p1 also points to y  */
```



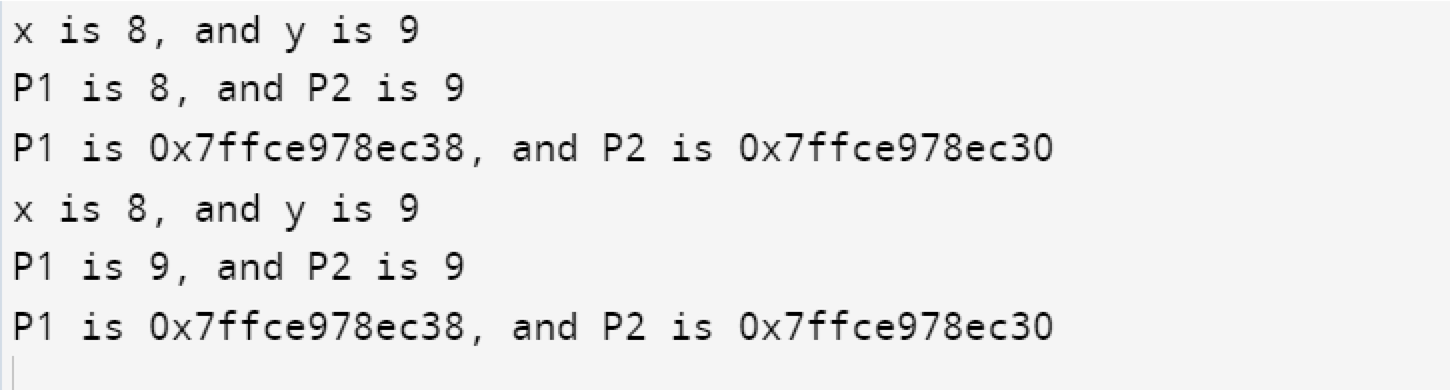
Assume x is at address 100, y is at address 800

Java:

```
Student s1 = new Student("John", 22);
Student s2 = new Student("Gorge",20);
s1 = s2;
```



```c
#include <stdio.h>

int main() {

int *p1, *p2;
int x = 8, y = 9;
p1 = &x; p2 = &y;

  printf("x is %d, and y is %d\n", x,y);
  printf("P1 is %d, and P2 is %d\n", *p1,*p2);
  printf("P1 is %p, and P2 is %p\n", &p1,&p2);
  p1= p2;

   printf("x is %d, and y is %d\n", x,y);
   printf("P1 is %d, and P2 is %d\n", *p1,*p2);
   printf("P1 is %p, and P2 is %p\n", &p1,&p2);
    return 0;
}
```

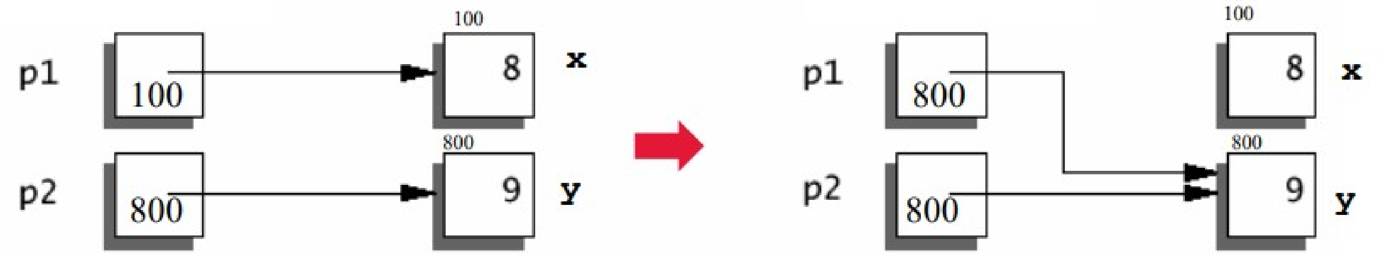


```
x is 8, and y is 9
P1 is 8, and P2 is 9
P1 is 0x7ffce978ec38, and P2 is 0x7ffce978ec30
x is 8, and y is 9
P1 is 9, and P2 is 9
P1 is 0x7ffce978ec38, and P2 is 0x7ffce978ec30
```

Some examples of Pointers -- summary

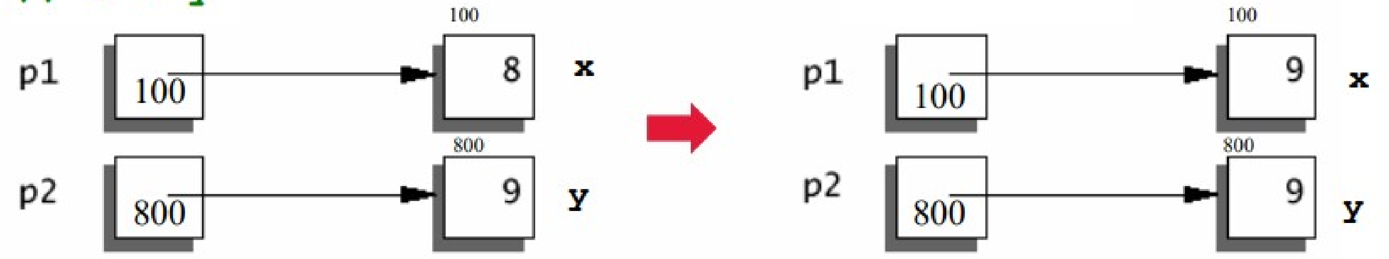
```
int *p1, *p2, x = 8, y = 9;

p1 = &x;  p2 = &y;
```

```
p1 = p2; // p1 = &y
```



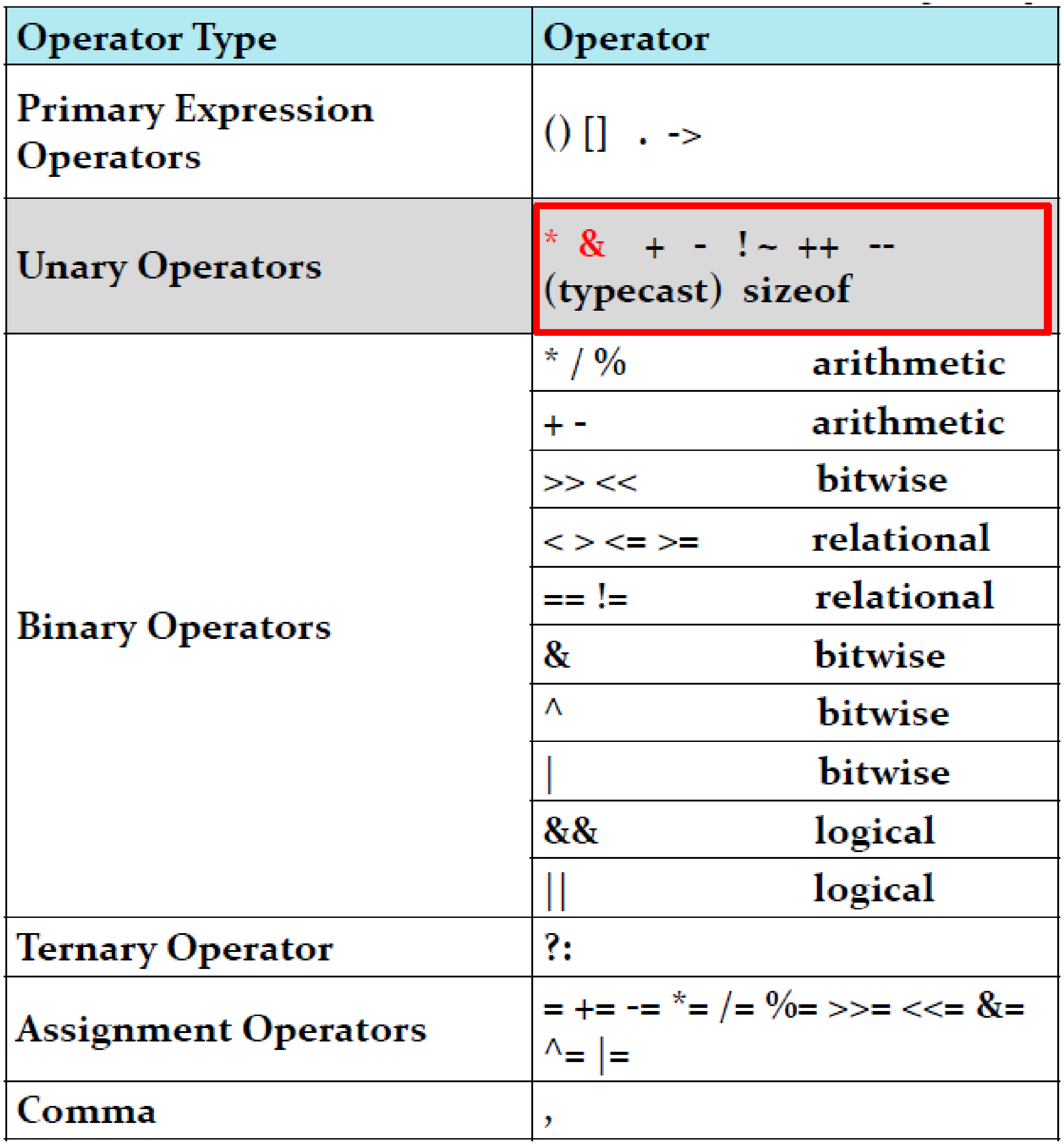
```
printf("%d %d\n", *p1, *p2); // 9 9
printf("%p %p\n",  p1, p2); //  800 800
```

```
*p1 = *p2; // x = y
```



```
printf("%d %d\n", *p1, *p2); // 9 9
printf("%p %p\n",  p1, p2); //  100 800
```

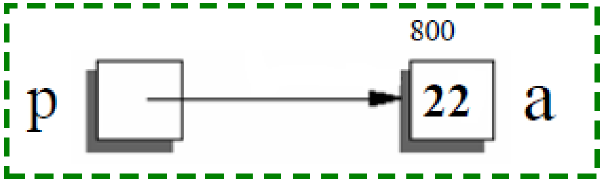
Precedence and Associativity



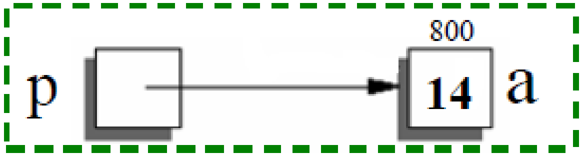
Operator Type	Operator	
Primary Expression Operators	() [] . ->	
Unary Operators	* & + - ! ~ ++ -- (typecast) sizeof	
Binary Operators	* / %	arithmetic
	+ -	arithmetic
	>> <<	bitwise
	< > <= >=	relational
	== !=	relational
	&	bitwise
	^	bitwise
	|	bitwise
	&&	logical
	||	logical
Ternary Operator	?:	
Assignment Operators	= += -= *= /= %= >>= <<= &= ^= |=	
Comma	,	

```
ptr = &x;
*ptr = 5;

y= *ptr + 4

ptr= &arr[0]
```

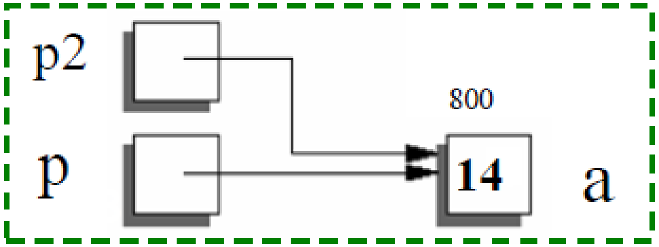
```
int main()
{
   int a = 22;
   int *p = &a;
   printf("%d %d\n", a, *p);   /* 22 22 */
```



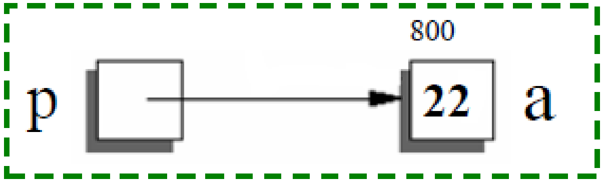
```
   *p =  14;   // a = 14
   printf("%d %d\n", a, *p);   /* 14 14 */
```

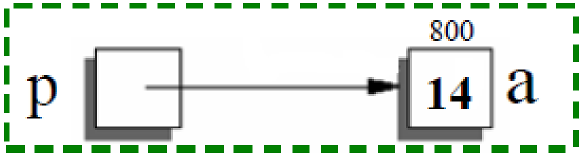


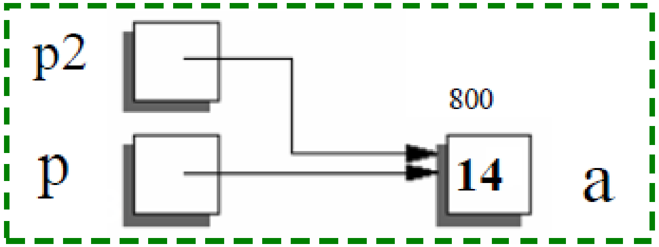
```
   int *p2 = p;


   (*p2)--;   // *p2 = *p2 - 1;
   printf("%d %d %d\n", a, *p, *p2);
   printf("%p %p %p\n", &a, p, p2);
```



```
int main()
{
  int a = 22;
  int *p = &a;
  printf("%d %d\n", a, *p);  /* 22 22 */

  *p =  14;  // a = 14
  printf("%d %d\n", a, *p);  /* 14 14 */

  int *p2 = p;

  (*p2)--;  // *p2 = *p2 - 1;
  printf("%d %d %d\n", a, *p, *p2);
  printf("%p %p %p\n", &a, p, p2);
```

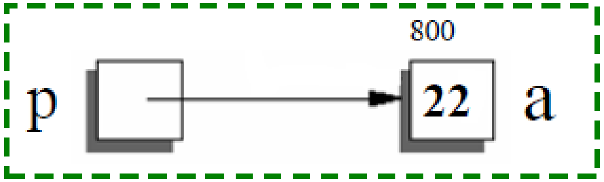




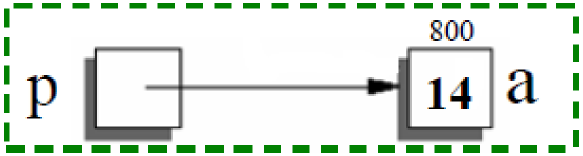


```
13 13 13
0x7ffc7e06dbcc 0x7ffc7e06dbcc 0x7ffc7e06dbcc
```

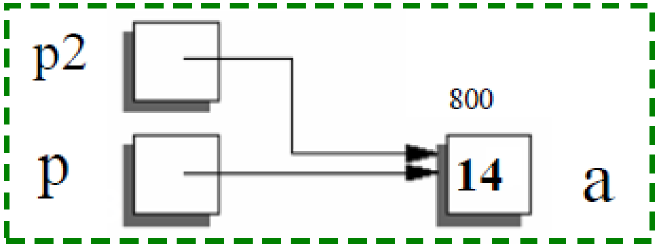
```
int main()
{
   int a = 22;
   int *p = &a;
   printf("%d %d\n", a, *p);  /* 22 22 */
```



```
   *p =  14;  // a = 14
   printf("%d %d\n", a, *p);  /* 14 14 */
```

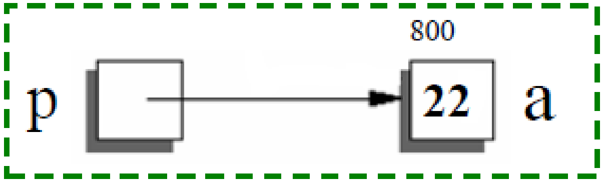


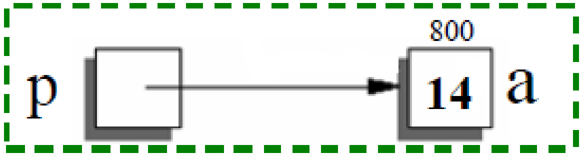
```
   int *p2 = p;

   (*p2)--;  // *p2 = *p2 - 1;
   printf("%d %d %d\n", a, *p, *p2);
   printf("%p %p %p\n", &a, p, p2);
```

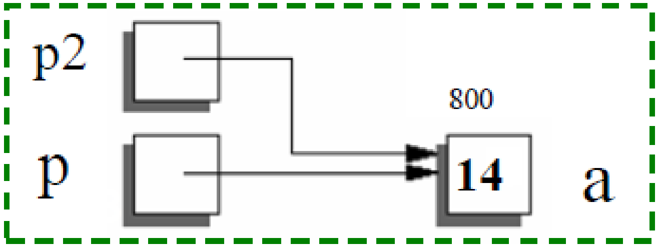


```
   double d = 23.32;
   int *p3 = &d;  ???
   double * p3 = &a; ???
```

```
int main()
{
  int a = 22;
  int *p = &a;
  printf("%d %d\n", a, *p);  /* 22 22 */

  *p =  14;  // a = 14
  printf("%d %d\n", a, *p);  /* 14 14 */

  int *p2 = p;

  (*p2)--;  // *p2 = *p2 - 1;
  printf("%d %d %d\n", a, *p, *p2);
  printf("%p %p %p\n", &a, p, p2);
```





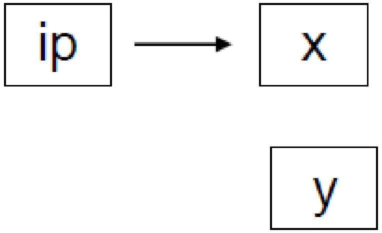


```
  double d = 23.32;
  int *p3 = &d;  ???
  double * p3 = &a; ???
```

Not valid! Type must match

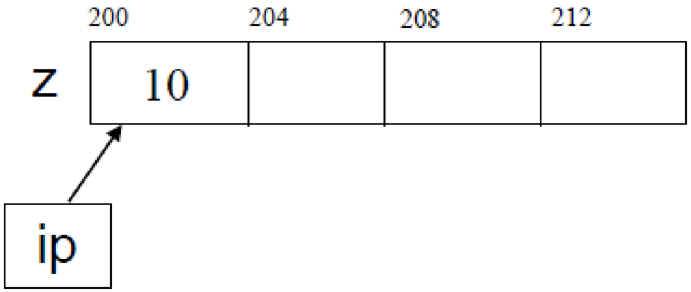
Another example

```
int x = 1, y = 2, z[4], k;
int *ip;
ip = &x;             /* ip points to x */

y = *ip;             /* y = x   y is now 1 */
*ip = 0;             /* x is now 0, y? */
```



Another example

```
z[0] = 10;
ip = &z[0];        /* ip points to z[0] now. *ip alise */
for (k = 1; k < 4; k++)
   z[k] = *ip + k;

*ip += 100; // *ip = *ip + 100
            // z[0] = z[0]+100
(*ip)++;
```



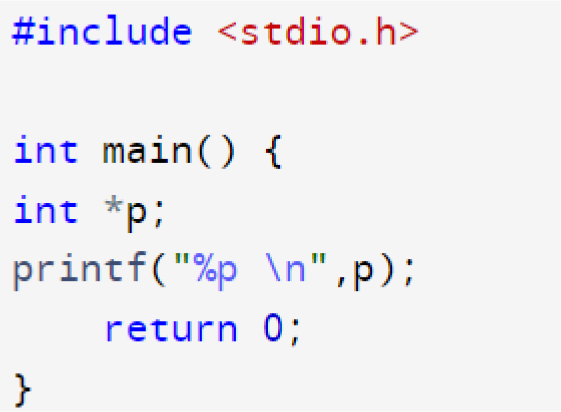
z: ? ? ? ?

NULL

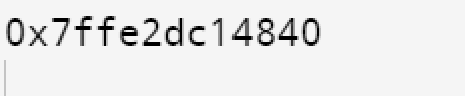
NULL: The Null Pointer is the pointer that does not point to any location but NULL.

Examples:

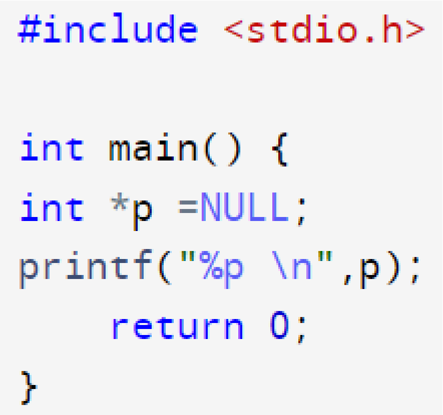
int *P = NULL;



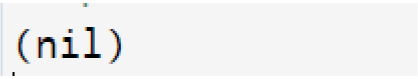
```
#include <stdio.h>

int main() {
int *p;
printf("%p \n",p);
    return 0;
}
```



```
0x7ffe2dc14840
```



```
#include <stdio.h>

int main() {
int *p =NULL;
printf("%p \n",p);
    return 0;
}
```



```
(nil)
```

When to use NULL

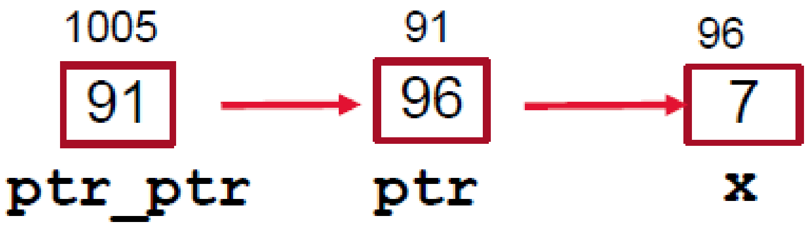
1. To initialize a pointer variable when that pointer variable hasn’t been assigned any valid memory address yet.

2. To check for a null pointer before accessing any pointer variable. By doing so, we can perform error handling in pointer-related code, e.g., dereference a pointer variable only if it’s not NULL.

3. To pass a null pointer to a function argument when we don’t want to pass any valid memory address.

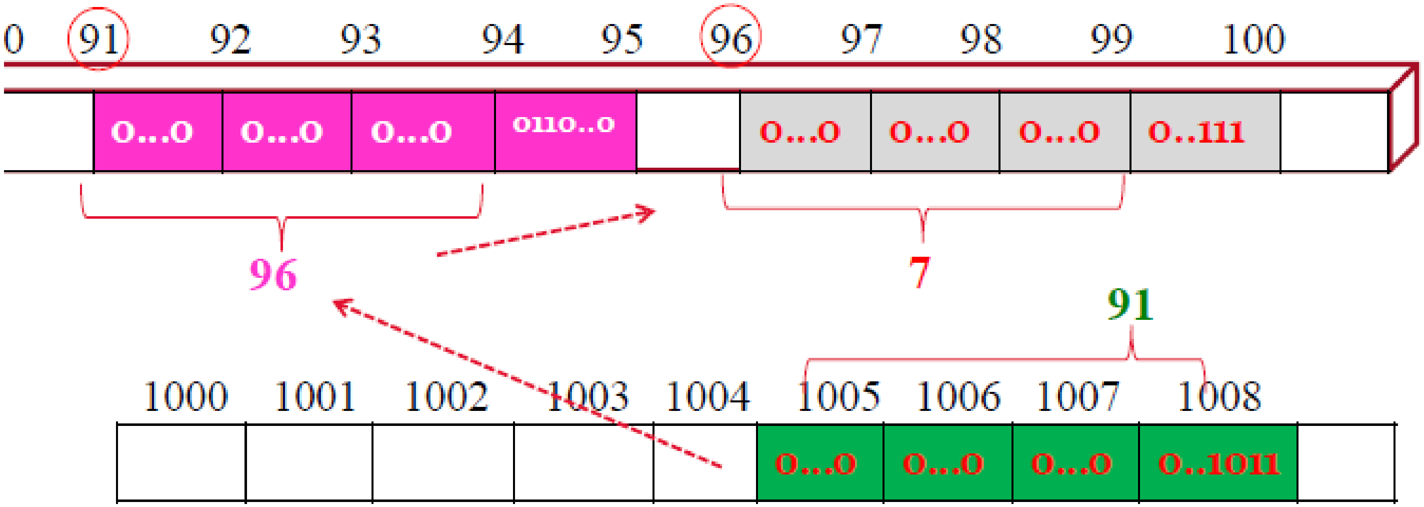
4. A NULL pointer is used in data structures like trees, linked lists, etc. to indicate the end.

Pointer to pointers

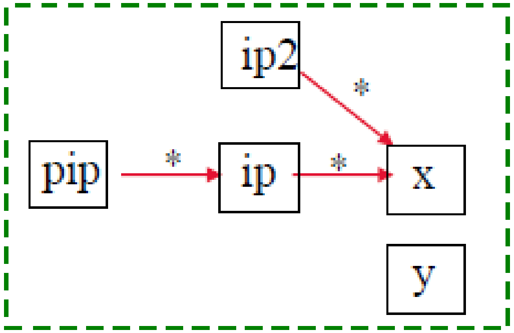


```
int x = 7;

int * ptr = &x;
```

```
int ** ptr_ptr      mnemonic:  // a pointer to pointer
ptr_ptr = &ptr;                // ptr_ptr value is 91

** ptr_ptr = 20;               // ** access x, set x to 20
```



More Examples



```
int x = 1, y = 2;
int *ip, *ip2;

ip = &x;

int **pip;      // I am a pointer to pointer
pip = &ip;     // pip points to pointer ip

y = **pip;
(**pip)--;

ip2 = ip;
*ip2 += 10;

ip = &y;
(**pip)--;

printf("%d %d\n", x,  y);
```

?