1806ICT
Programming Fundamentals

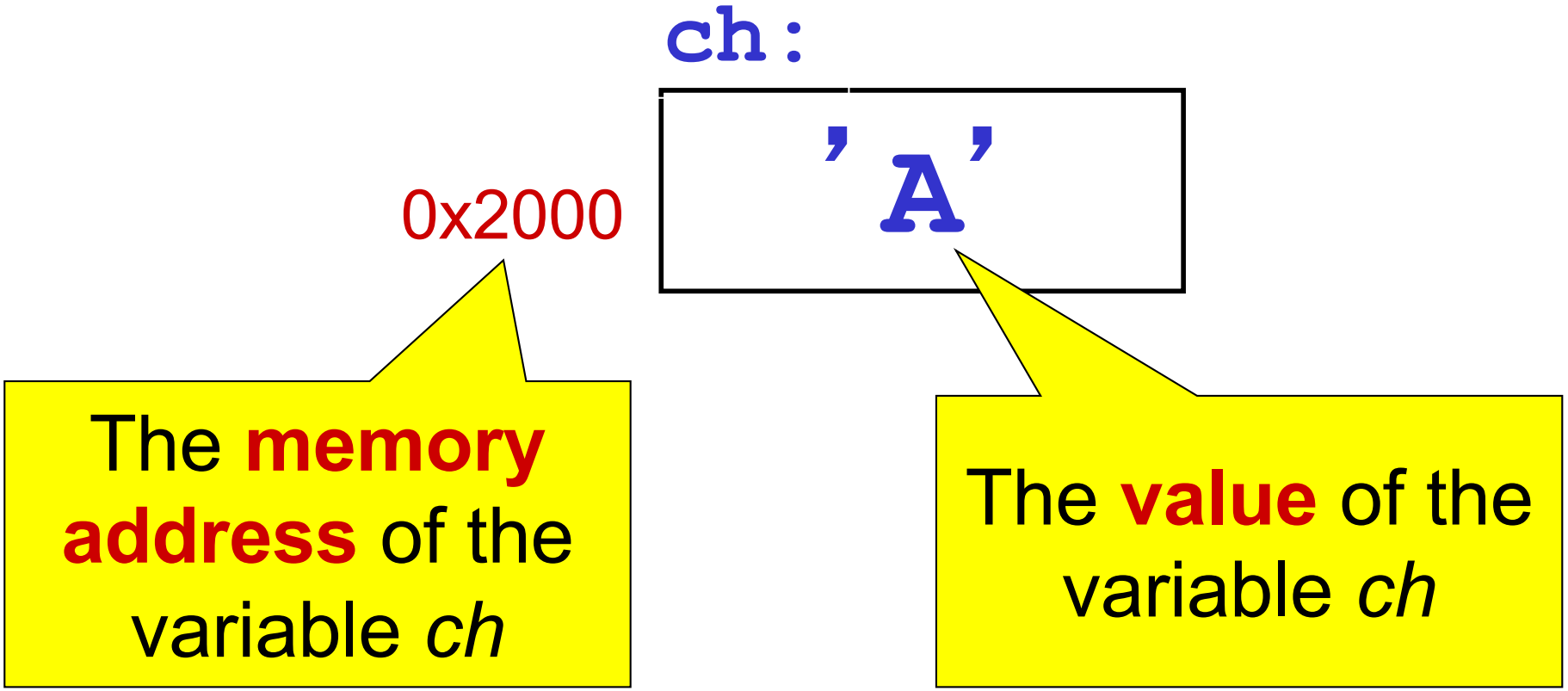
C Pointers

Topics

- Introduction to Pointers
- Pointers and Function Parameters
- Pointers and Arrays

Memory Address of a Variable

```
char ch = 'A';
```



The & Operator

- Gives the memory address of an object

```
char ch = 'A';
```

0x2000 | 'A'

`&ch` yields the value 0x2000

- Also known as the "address operator"

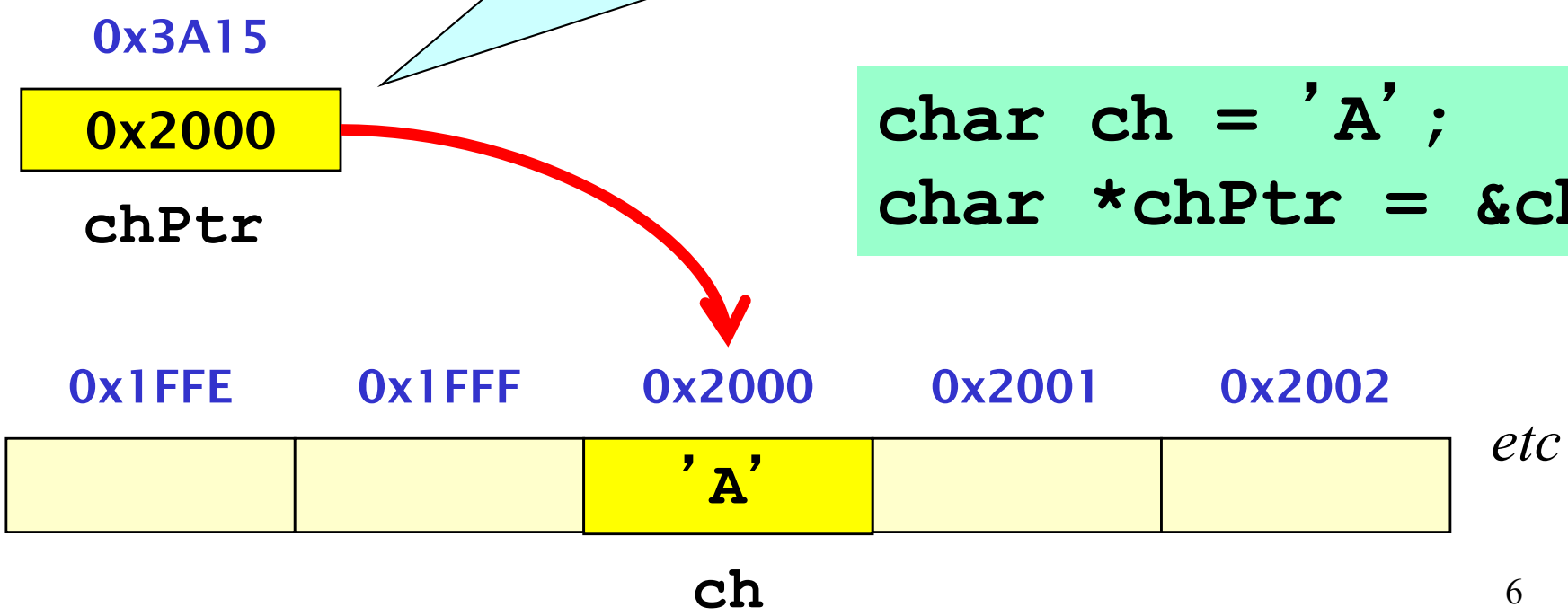
Example:

```
char ch;
printf("%p", &ch);
```

"conversion specifier" for printing a memory address

Pointers

A variable which can store the **memory address** of another variable




```
char ch = 'A';
char *chPtr = &ch;
```

Pointers

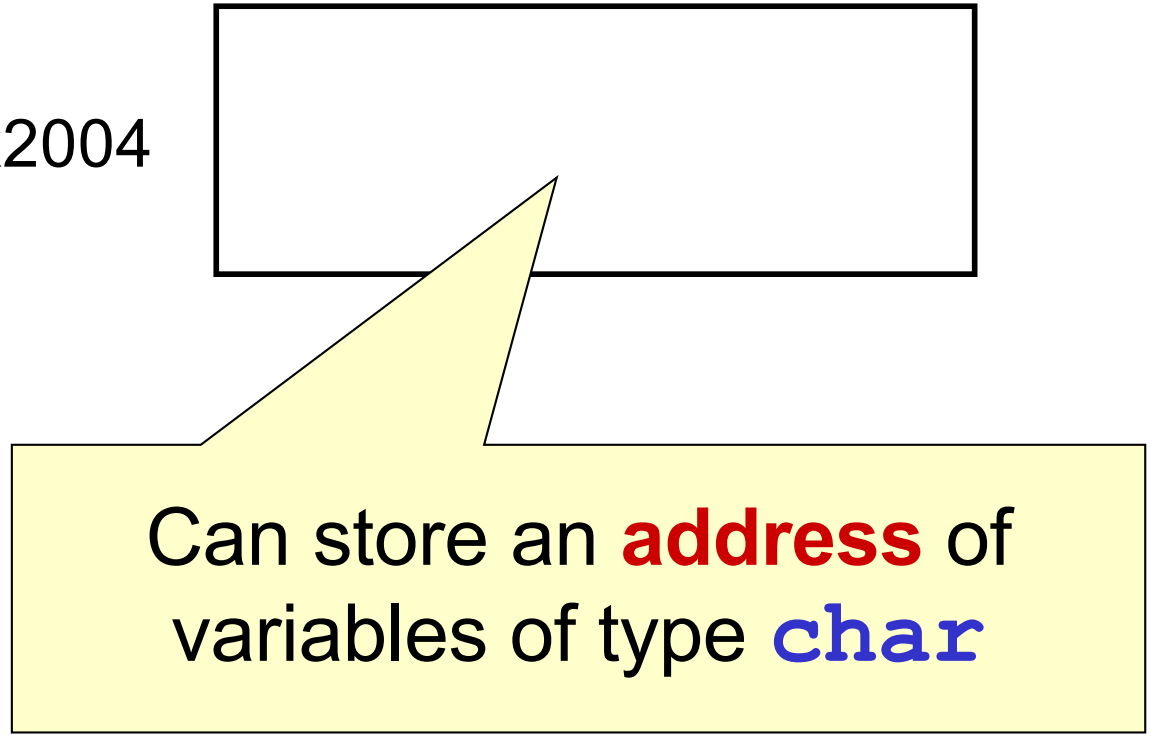
- A pointer is a variable
- Contains a memory address
- Points to a specific data type
- Pointer variables are usually named ***`varPtr`***

Example:

```
char *cPtr;
```

`cPtr:`

0x2004



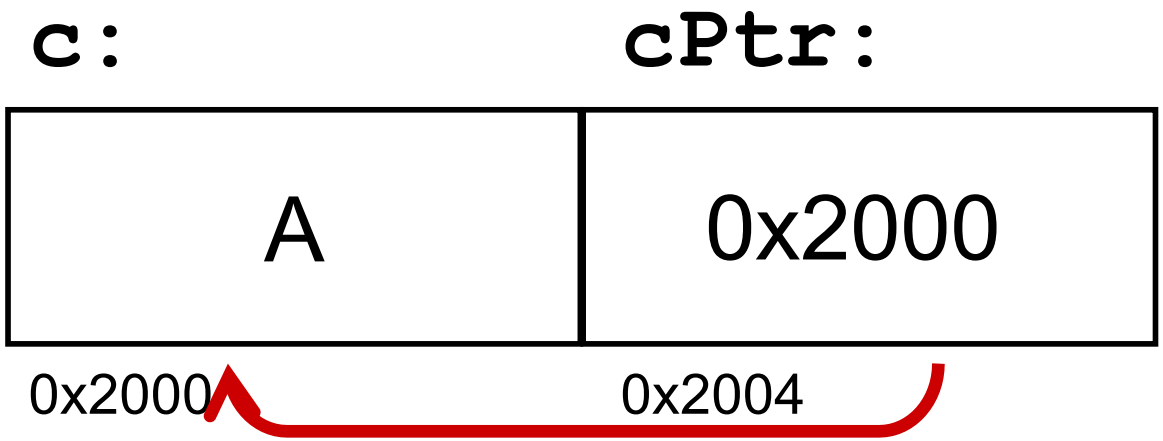
- We say *cPtr* is a ***pointer*** to char

Pointers and the & Operator

Example:

```
char c = 'A';
char *cPtr;

cPtr = &c;
```

Assigns the address of c to cPtr



Notes on Pointers

- We can have pointers to any data type

Example:

```
int*    numPtr;
float*   xPtr;
```

- The * can be anywhere between the type and the variable

Example:

```
int     *numPtr;
float * xPtr;
```

Notes on Pointers (cont)

- You can assign the address of a variable to a "compatible" pointer using the **&** operator

Example:

```
int   aNumber;
int   *numPtr;

numPtr = &aNumber;
```

- You can print the address stored in a pointer using the **%p** conversion specifier

Example:

```
printf("%p", numPtr);
```

The * Operator

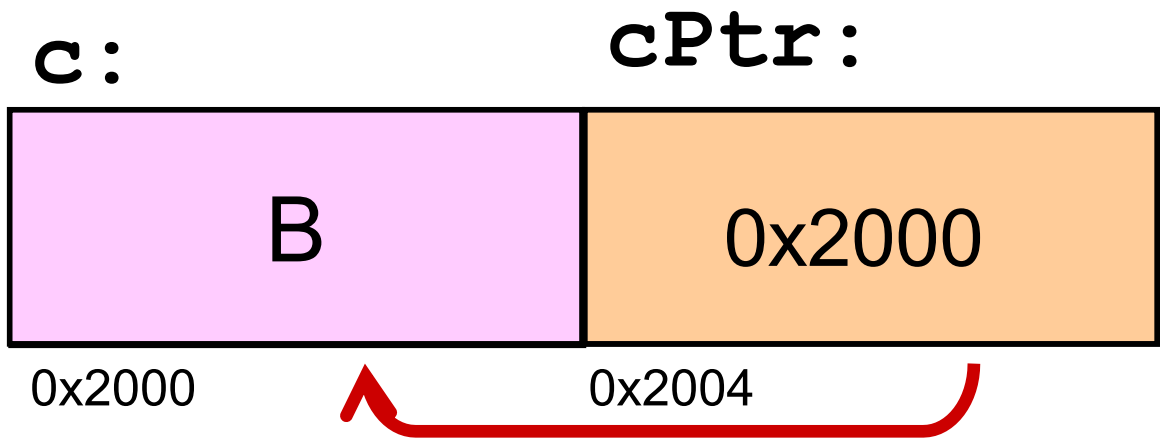
- Allows pointers to access variables they point to
- Also known as "dereferencing operator"
- Should not be confused with the * in the pointer declaration

Pointers and the * Operator

Example:

```
char c = 'A';
char *cPtr = NULL;


cPtr = &c;
*cPtr = 'B';
```

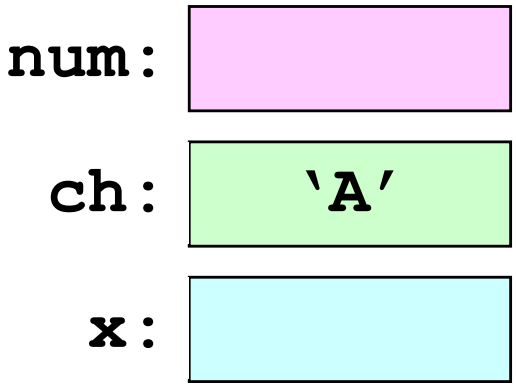
Changes the value of the variable which cPtr points to



Easy Steps to Pointers

- ***Step 1***: Declare the variable to be pointed to

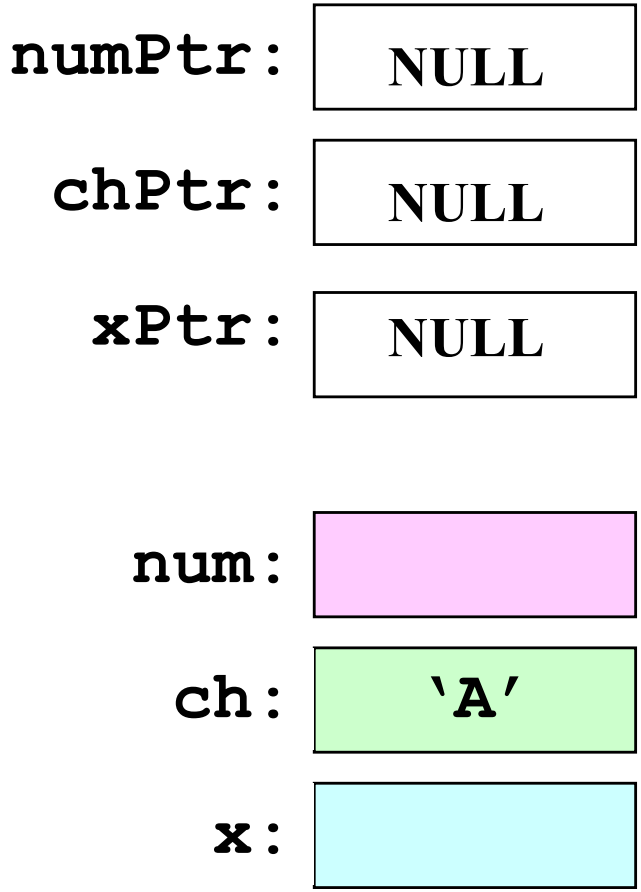
```
int  num;
char ch = 'A';
float x;
```



Easy Steps to Pointers (cont)

- ***Step 2***: Declare the pointer variable

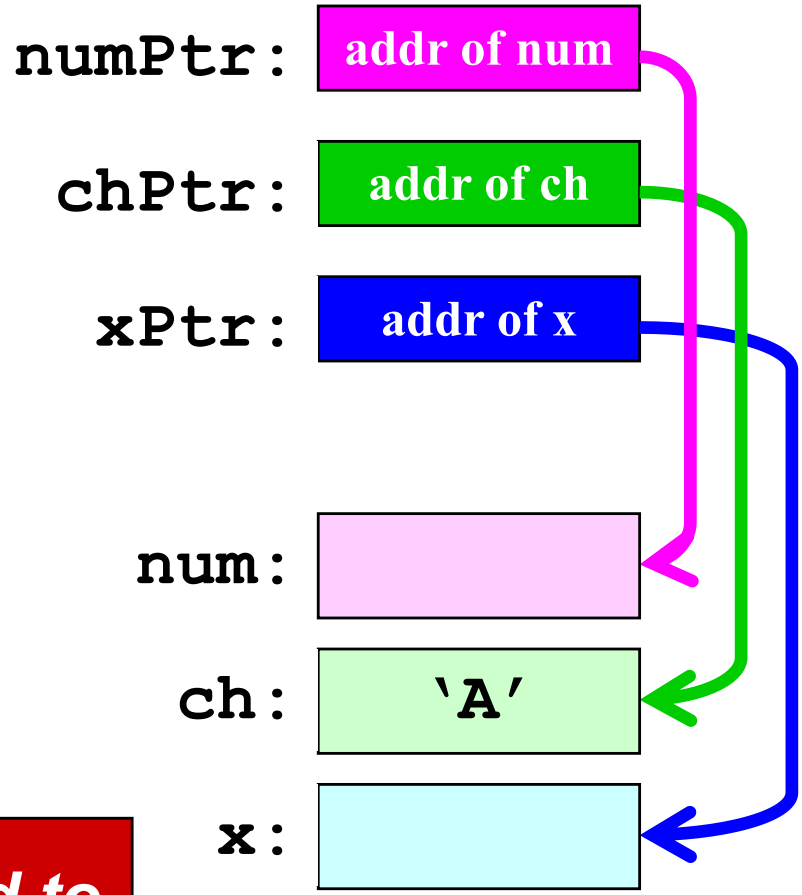
```
int  num;
char ch = 'A';
float x;

int*  numPtr = NULL;
char  *chPtr = NULL;
float * xPtr = NULL;
```



Easy Steps to Pointers (cont)

- ***Step 3***: Assign address of variable to pointer

```
int   num;
char  ch = 'A';
float x;

int*  numPtr = NULL;
char  *chPtr = NULL;
float * xPtr = NULL;

numPtr = &num;
chPtr = &ch;
xPtr = &x;
```

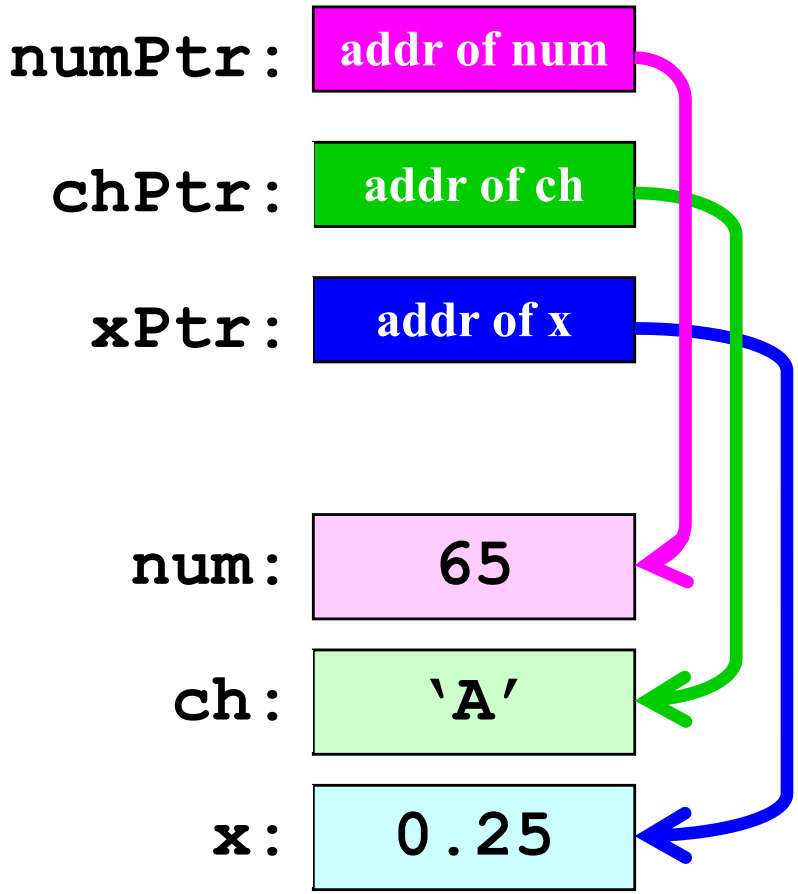


A pointer's type has to correspond to the type of the variable it points to

Easy Steps to Pointers (cont)

- ***Step 4***: De-reference the pointers

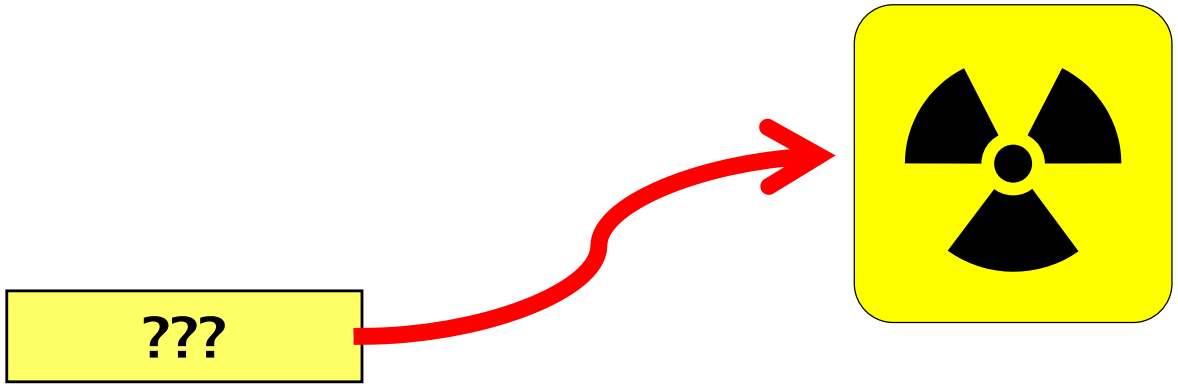
```
int   num;
char  ch = 'A';
float x;

int*  numPtr = NULL;
char  *chPtr = NULL;
float * xPtr = NULL;

numPtr = &num;
chPtr = &ch;
xPtr = &x;

*xPtr = 0.25;
*numPtr = *chPtr;
```



Notes on Pointers (cont)

```
int   *numPtr;
```

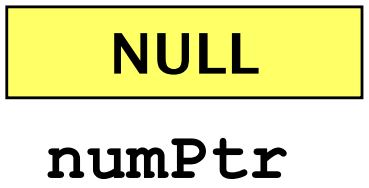
Beware of pointers which are not initialized!



Notes on Pointers (cont)

- When declaring a pointer, it is a good idea to always initialize it to **NULL** (a special pointer constant)

```
int  *numPtr = NULL;
```

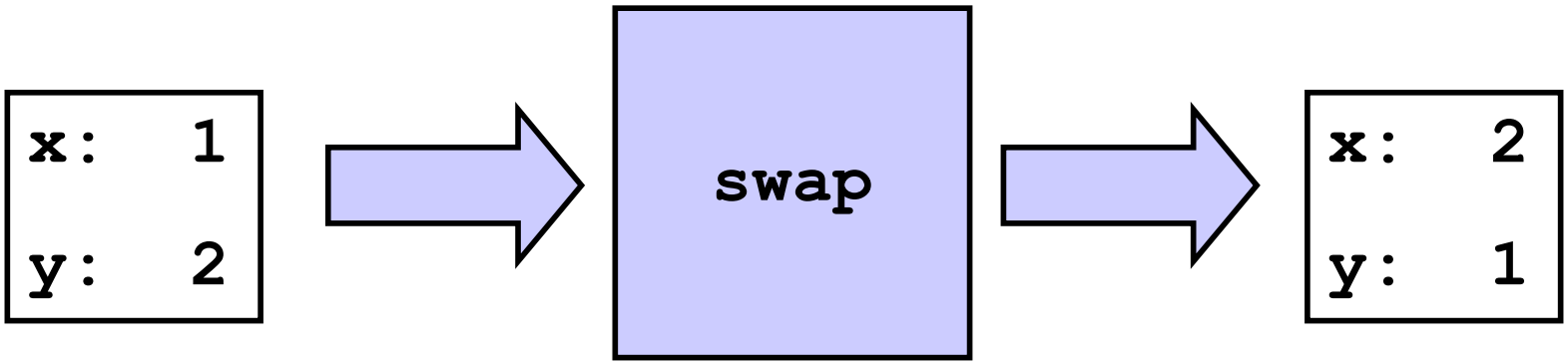


Topics

- ✓ Introduction to Pointers
- Pointers and Function Parameters
- Pointers and Arrays

Pointers and Function Parameters

- ***Example:*** Function to swap the values of two variables

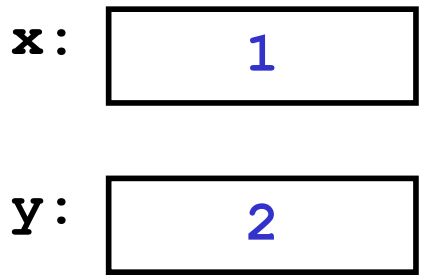


Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```

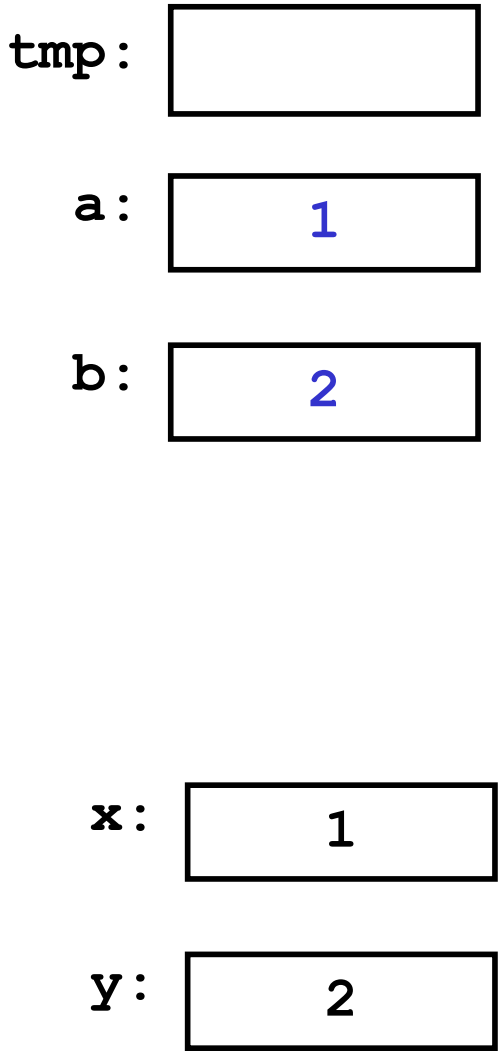
Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```



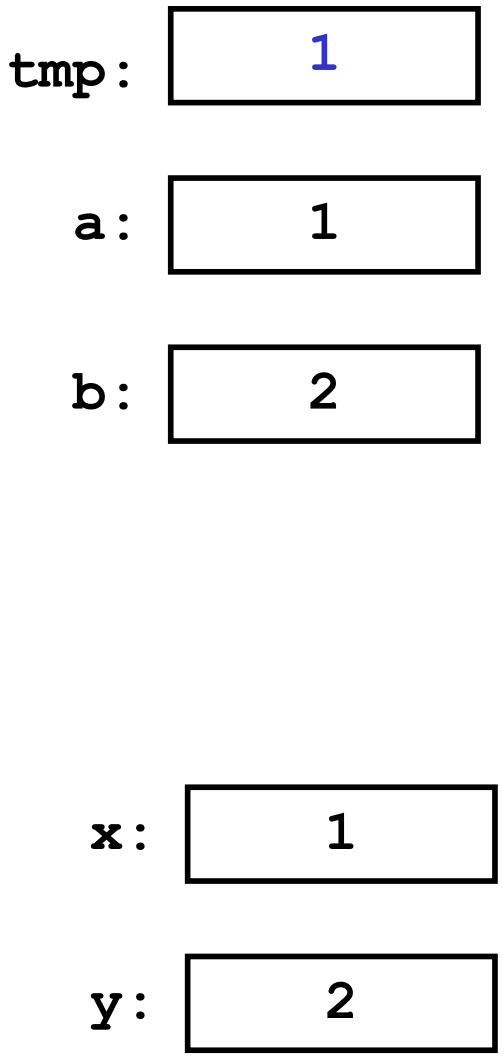
Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```



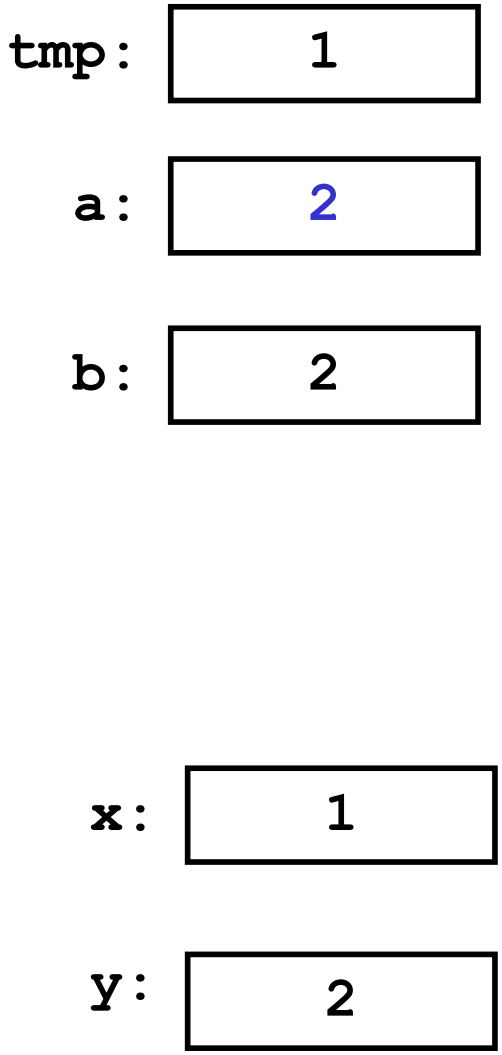
Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```



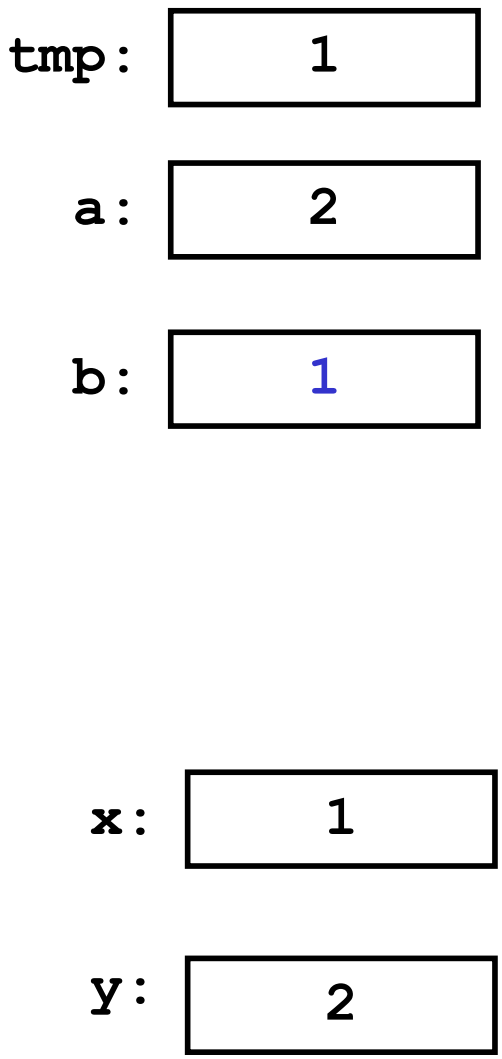
Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```



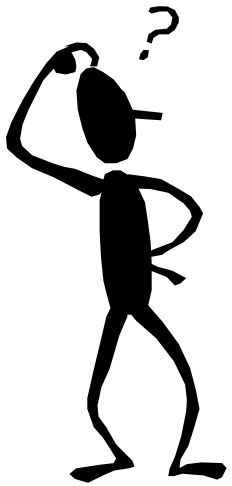
Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```



Bad swap

```
#include <stdio.h>

void swap1(int a, int b)
{
    int tmp;

    tmp = a;
    a = b;
    b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap1(x, y);
    printf("%d %d\n", x, y);
    return 0;
}
```



tmp:	1
a:	2
b:	1

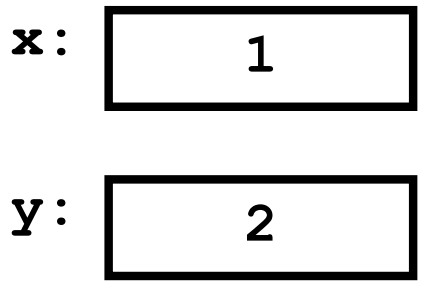
x:	1
y:	2

Good swap

```
#include <stdio.h>

void swap2(int* a, int* b)
{
   int tmp;

   tmp = *a;
   *a = *b;
   *b = tmp;
   return;
}

int main()
{
   int x = 1, y = 2;

   swap2(&x, &y);
   printf("%d %d\n", x, y);
   return 0;
}
```

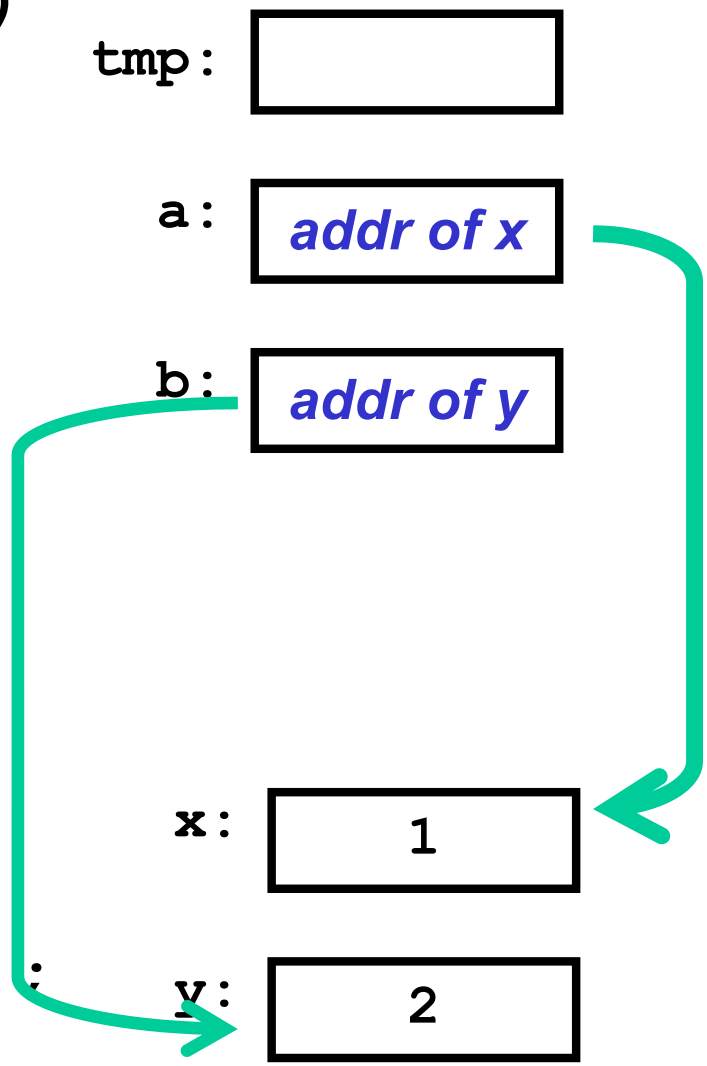
Good swap

```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```



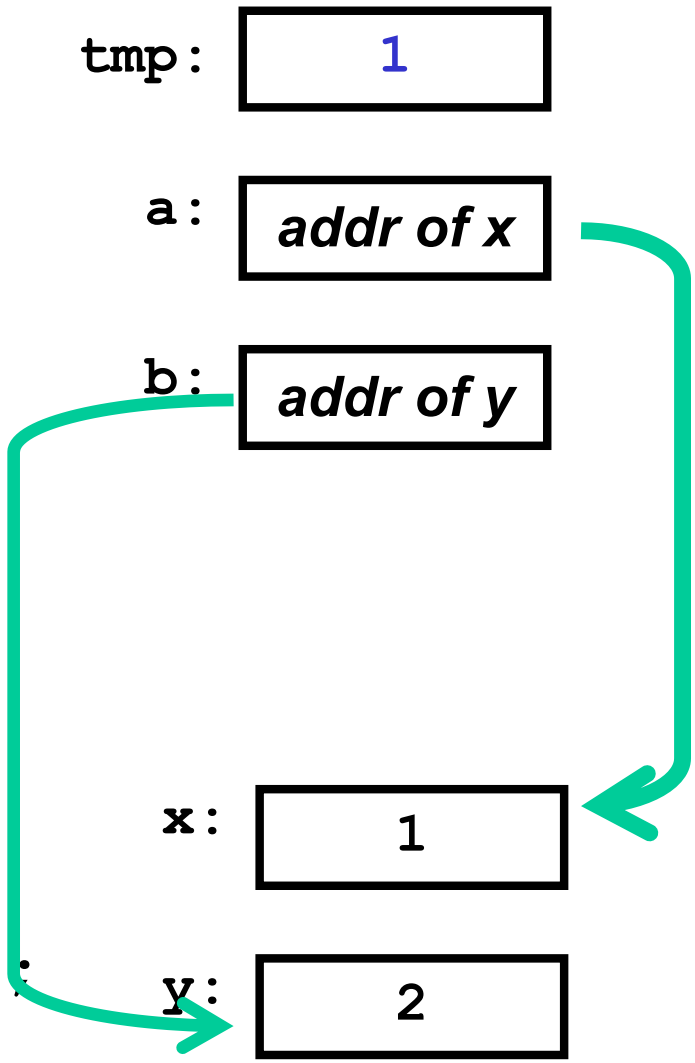
Good swap

```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```



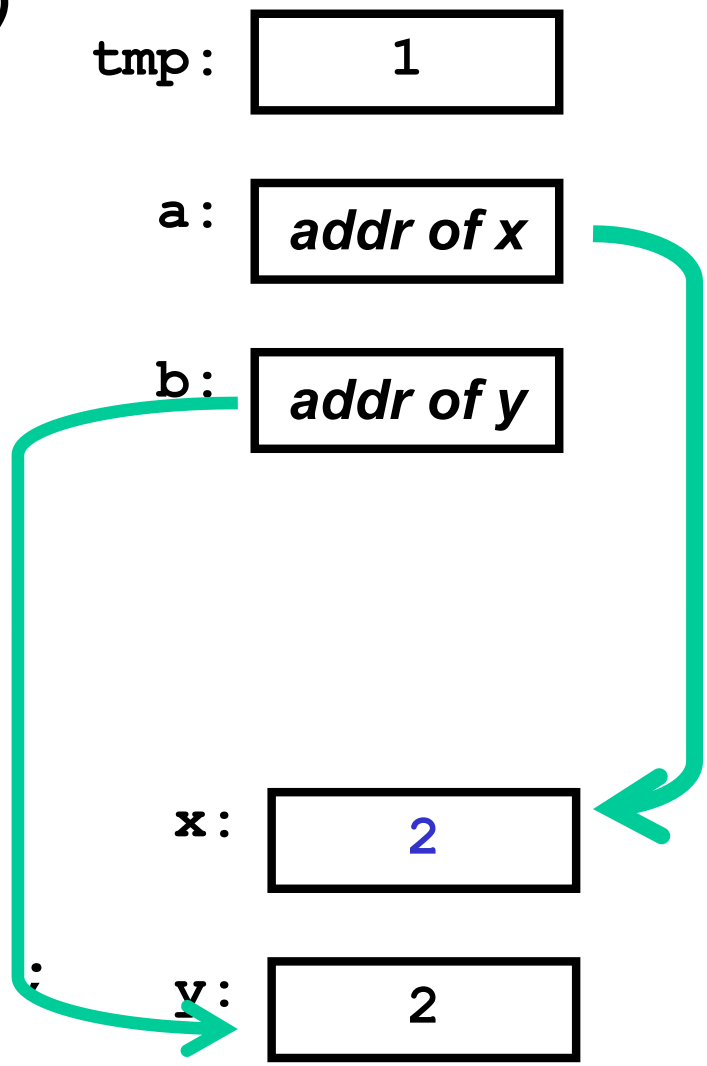
Good swap

```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```



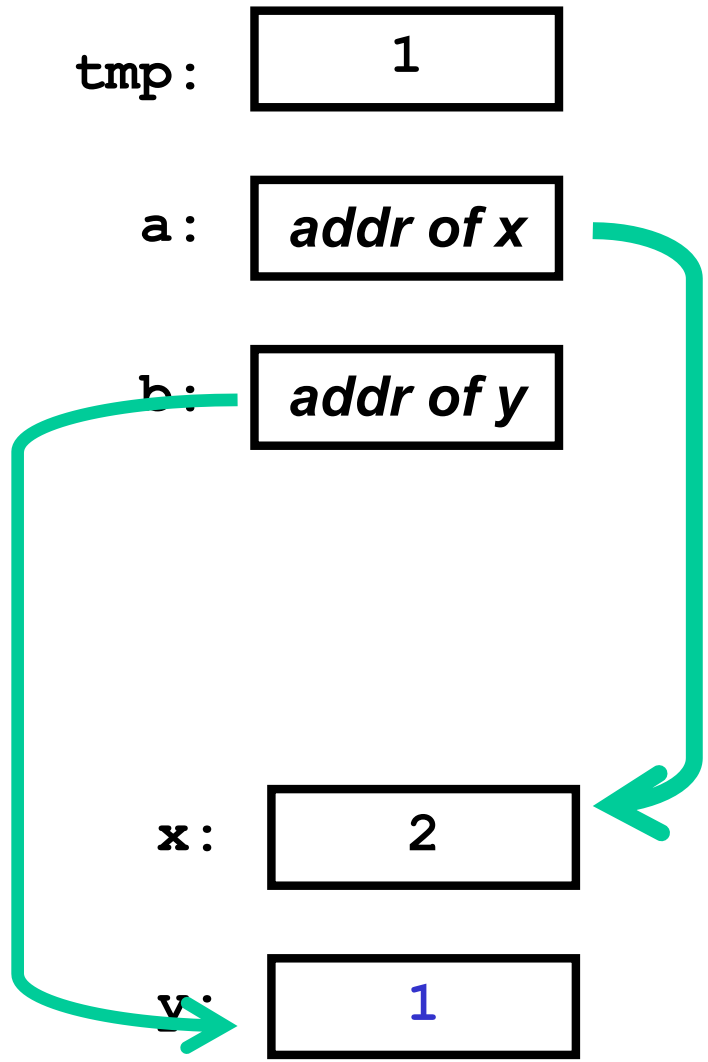
Good swap

```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```



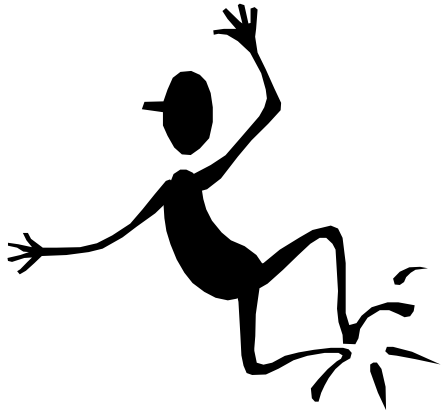
Good swap

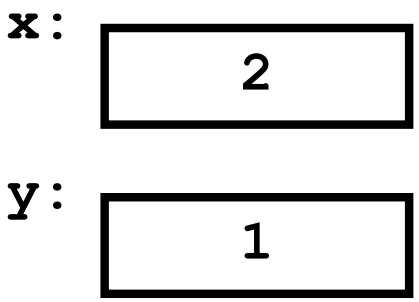
```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```



Good swap

```
#include <stdio.h>

void swap2(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}

int main()
{
    int x = 1, y = 2;

    swap2(&x, &y);
    printf("%d %d\n", x, y);
    return 0;
}
```





Pointers and Function Arguments

- Change the value of an actual parameter variable
- **`scanf`** demystified

```
char ch;

int  numx;

float numy;

scanf("%c %d %f", &ch, &numx, &numy);
```

Topics

- ✓ Introduction to Pointers
- ✓ Pointers and Function Parameters
- Pointers and Arrays

Pointers and Arrays

- An array name by itself is an address
- Therefore, an array name is a pointer

```
#include <stdio.h>

int main()
{
   int array[3] = {10, 20, 30};
   int *arrayPtr = NULL;

   arrayPtr = array;
   for (int i=0; i<3; i++)
   {
      printf("array[%d] = %d\n", i, array[i]);
      printf("array[%d] = %d\n", i, *(arrayPtr+i));
      printf("array[%d] = %d\n", i, *(array+i));
   }
   return 0;
}
```

These three statements are equivalent

Pointers and Arrays – Another Example

```
#include <stdio.h>

int main()
{
   int array[3] = {10, 20, 30};
   int *arrayPtr = NULL;

   arrayPtr = array + 1;    // pointer points to array[1]
   *arrayPtr = 70;          // change array[1] to 70

   for (int i=0; i<3; i++)
   {
      printf("array[%d] = %d\n", i, array[i]);
   }
   return 0;
}
```

Output:
array[0] = 10
array[1] = 70
array[2] = 30

Pointers and Arrays – modifyArray Example

```
#include <stdio.h>

/* multiply each element by 2 */
void modifyArray(int *numsPtr, int size)
{
   for (int i=0; i<size; i++)
      numsPtr[i] = numsPtr[i] * 2;
}

int main()
{
   int nums[3] = {10, 20, 30};

   modifyArray(nums, 3);

   for (int i=0; i<3; i++)
      printf("nums[%d] = %d\n", i, nums[i]);

   return 0;
}
```

Output:
nums[0] = 20
nums[1] = 40
nums[2] = 60

Pointers and Arrays – Differences

- An array name is an address, or pointer, that is **fixed**
- Therefore, an array name is not a variable

```
int array1[3] = {10, 20, 30};
int array2[3] = {1, 2, 3};

int *arrayPtr = NULL;

arrayPtr = array1;     // this is OK

arrayPtr = arrayPtr + 1;  // this is OK

array1 = array2;       // this is illegal. Compile error

array1 = array1 + 1;  // this is illegal. Compile error
```

Summary

- Introduction to Pointers
- Pointers and Function Parameters
- Pointers and Arrays