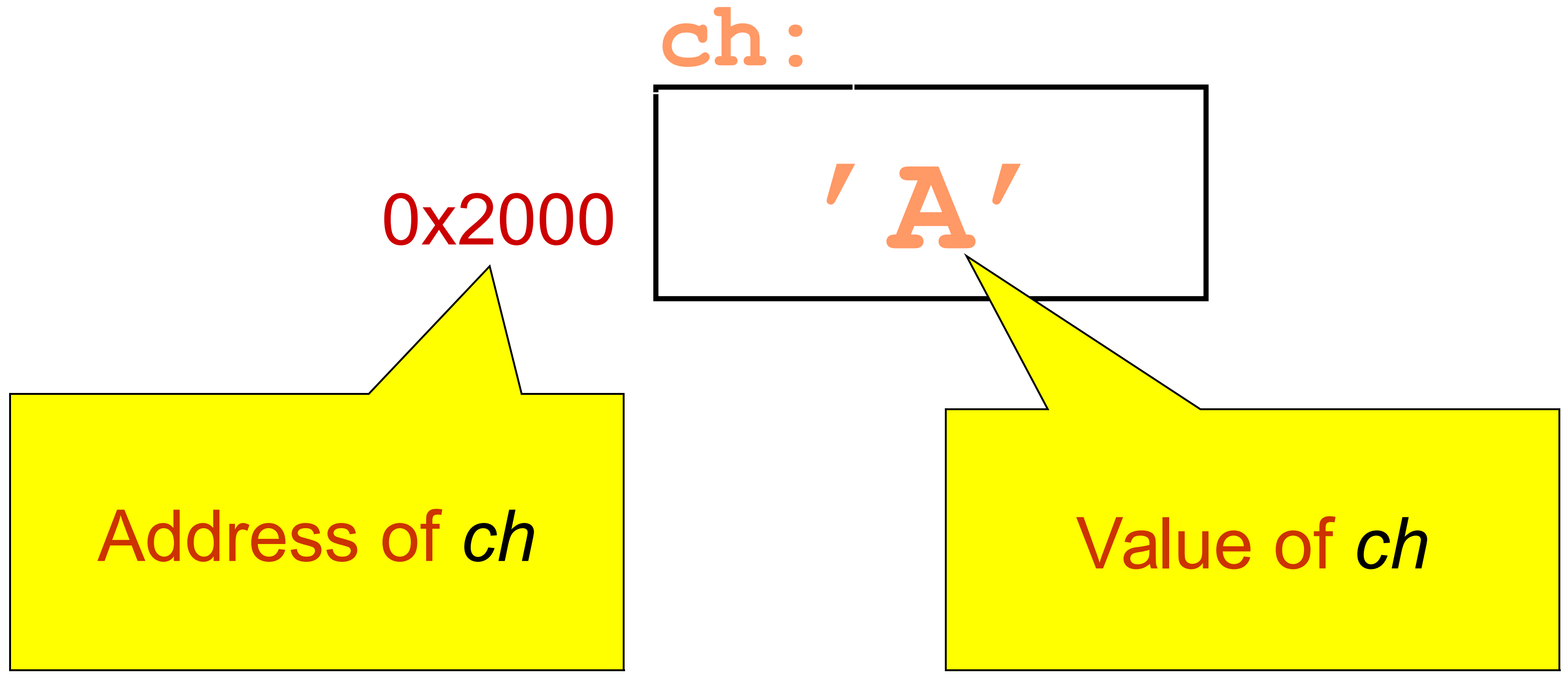
Pointer

Department of Information System

SoICT, HUST

Memory address of a variable

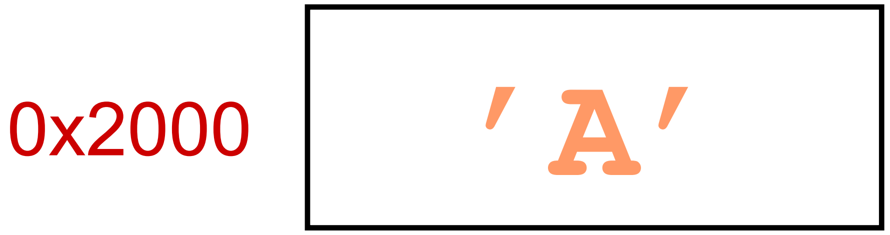
```
char ch = 'A';
```



Operator **&**

- Yields the memory address of an object

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

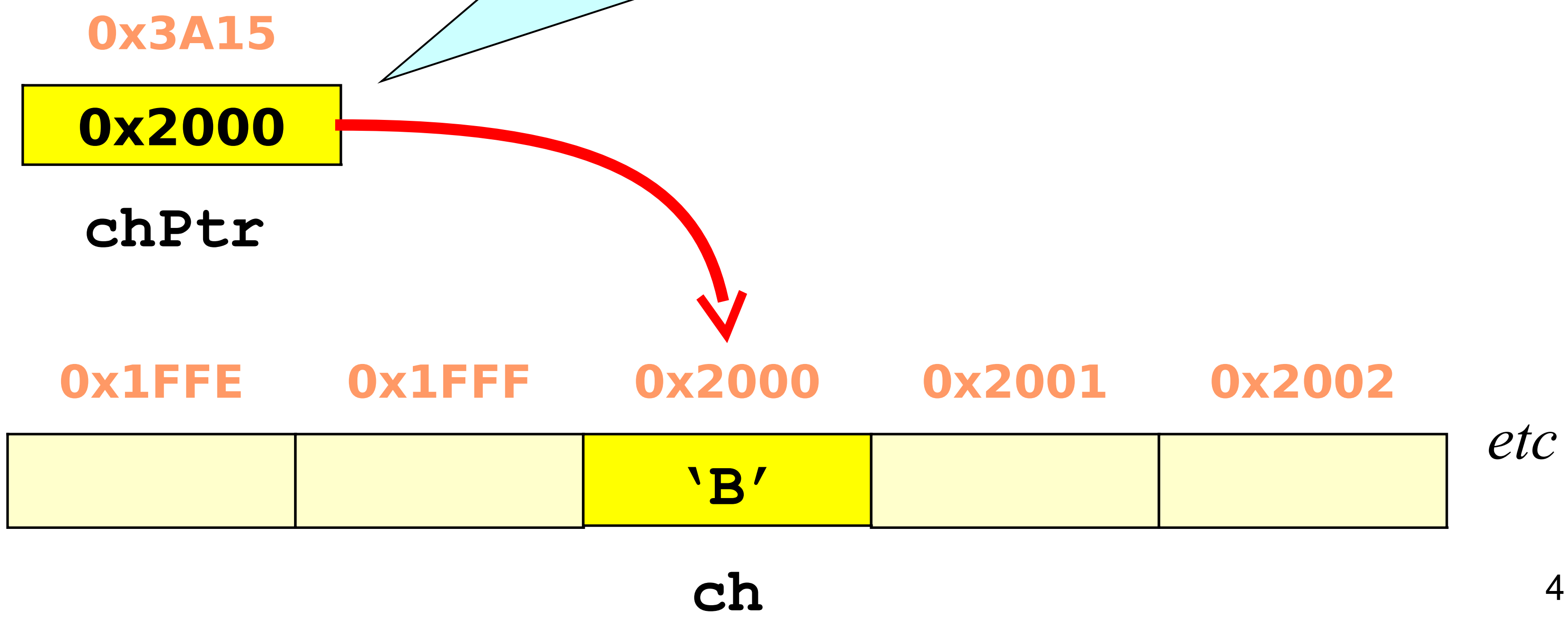


```
&ch
```

Return 0x2000

Pointer

A variable can store a value which is address of another variable

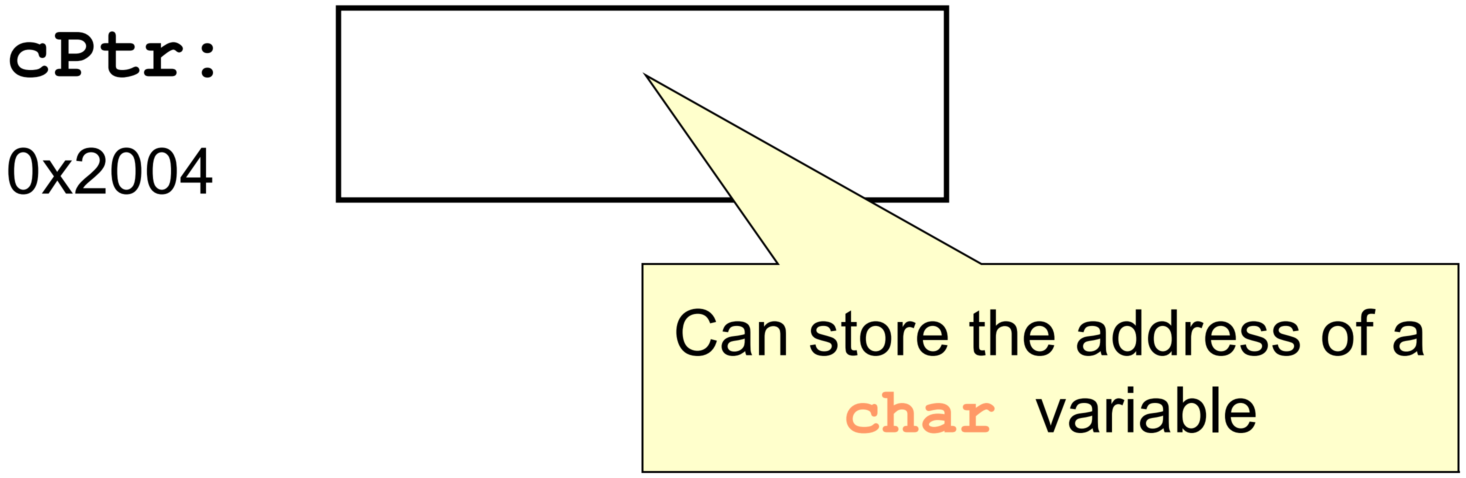


Pointer declaration

- A pointer is a variable which
  - Contains a memory address
  - Points to a specific data type

<u>Example:</u>

```
char* cPtr;
```



Pointer declaration

- can declare a pointer that points to any data type.

Example:

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

- A pointer variable is always declared with an operator *

Example:

```
int *numPtr1, *numPtr2;
float *xPtr, *yPtr;
```

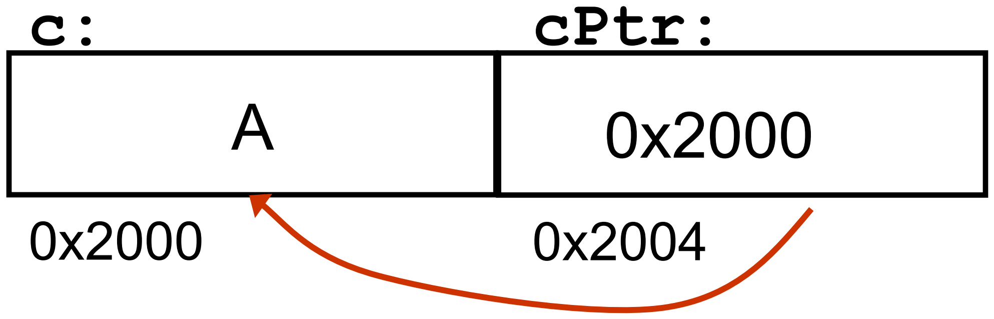
Dereferencing

- Operator & is used to get the reference address of a pointer

Example:

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

Assign the address of c to the pointer cPtr



Note

•The dereference variable of a pointer must have the corresponding type with the pointer.

Example:

```
int  aNumber;
char *ptr;

ptr = &aNumber;
```

Data type error

•To print the value of a pointer, we use the format **%p**

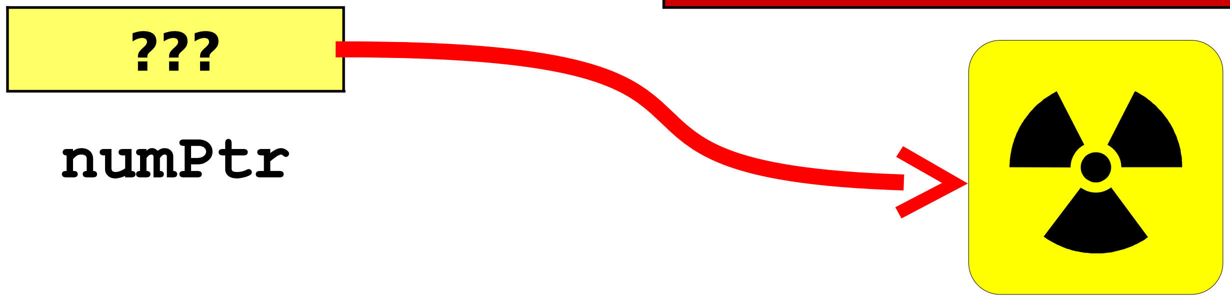
Example:

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

NULL pointer

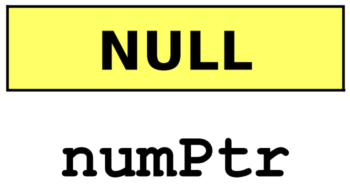
```
int  *numPtr;
```

Be careful with the pointer not initialized



- A pointer should be initialized before using. If there is not a variable to point to, initialize it with NULL (a special value = 0).

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



A pointer with the NULL dereference (no address)

* operator

- We can use pointers to access variables they point to by the * operator.
- * is also known as “dereferencing operator”.
- Should not be confused with the * in the pointer declaration.
- Be careful with the pointer not initialized

Example:

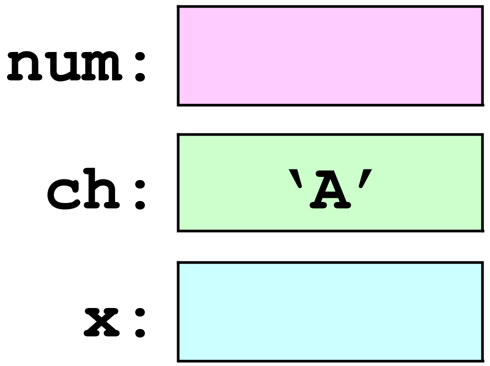
```
char c = 'A';
char *cPtr = NULL;
cPtr = &c;
*cPtr = 'B';
```

Change value of c pointed by cPtr

Steps in using a pointer

- ***Step 1***: Declare a variable pointed by a pointer

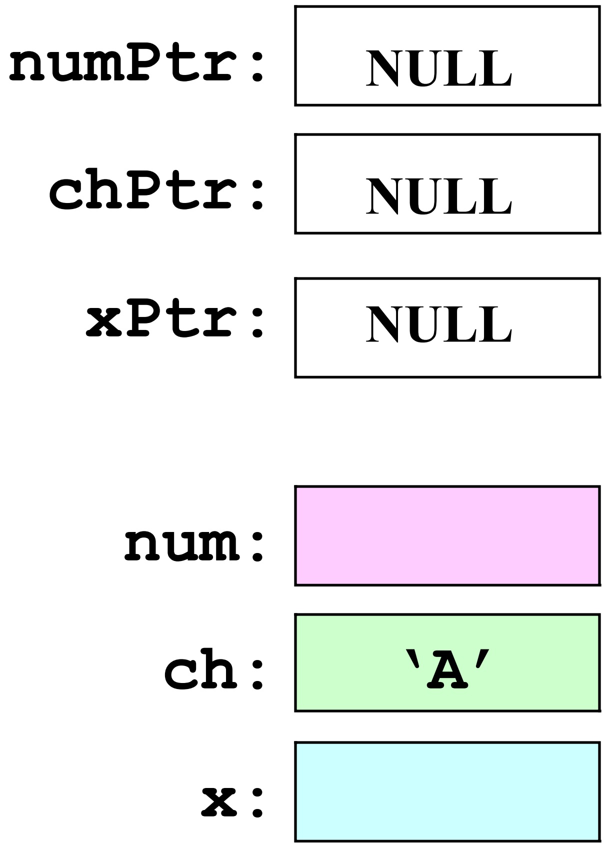
```
int  num;
char ch = 'A';
float x;
```



Steps in using a pointer

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

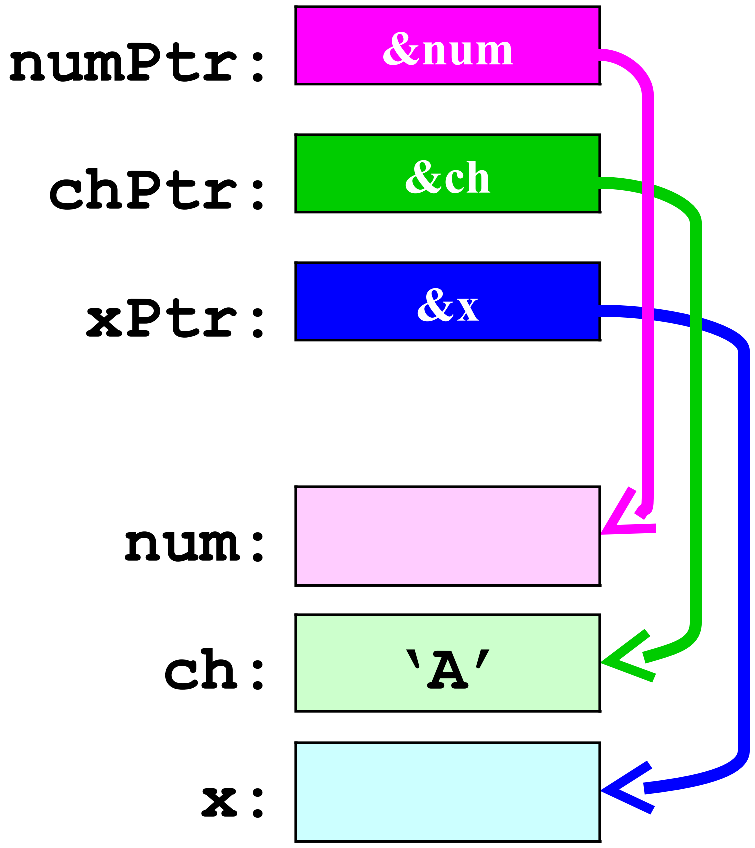
```
int   num;
char ch = 'A';
float x;

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



Steps in using a pointer

- ***Step 3:*** Referencing the pointer

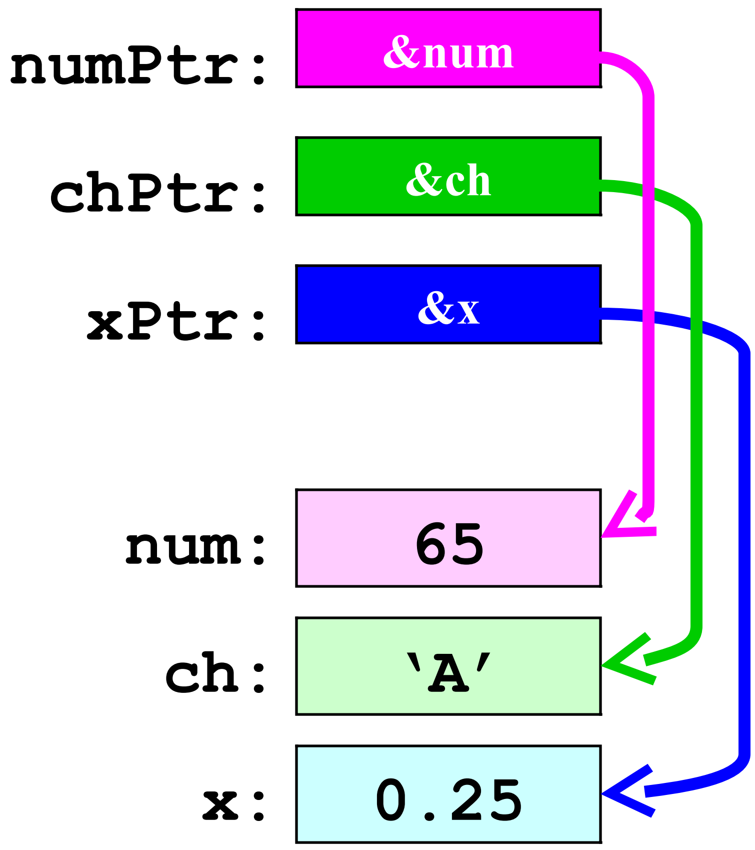
```
int    num;
char   ch = 'A';
float  x;

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

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



Steps in using a pointer

- ***Step 4***: Dereferencing the pointer

```
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;
```



Common errors

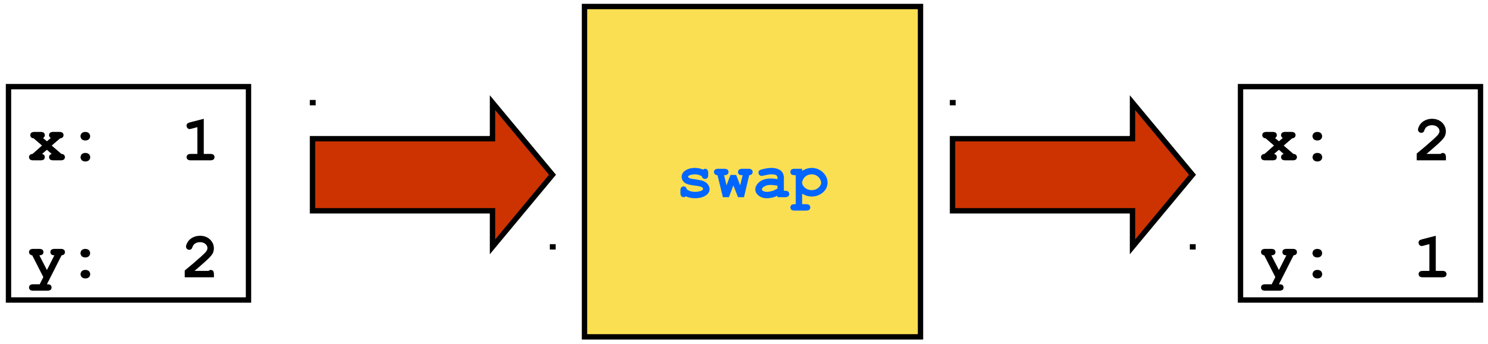
- Cannot referencing a pointer to a constant or an equation.
- Cannot change the address of a variable in the memory (since it cannot determine by users!)
- Errors:
  - ptr = &3;
  - ptr = &(x+5);
  - &x = ptr;
  - &x = 0x2000;

Example

```
int main()
{   int x = 25, y = 50;    // Two int variables
    int *ptr;                // Pointer variable
    // Display the contents of x and y.
   printf(“%d   %d”, x,y);
    // Use the pointer to manipulate x and y.
    // Store the address of x in ptr.
    ptr = &x;
    // Add 100 to the value in x.
    x +=100;    // *ptr = *ptr +100;
    // Store the address of y in ptr.
    ptr = &y;
    // Add 100 to the value in y.
    y +=100;   //*ptr = *ptr + 100;
    // Display the contents of x and y.
   printf(“%d   %d”, x,y);
    return 0;
}
```

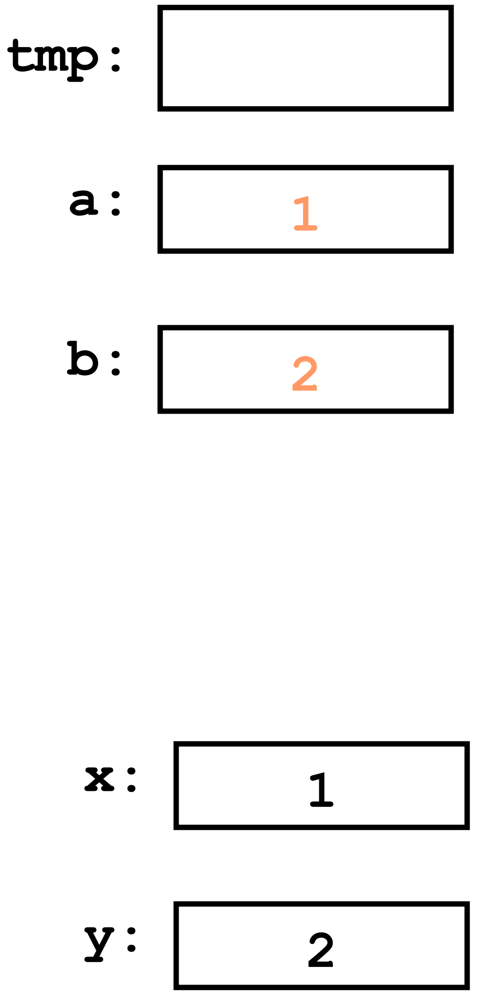
Function parameters and pointers

- *Pointers can be passed as parameters of a function.*
- *Example: Create a function that swap values of two input parameter.*



Passing parameters by values

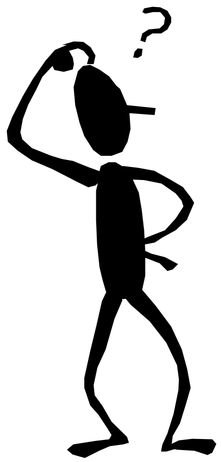
```
#include <stdio.h>

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;
}
```



Passing parameters by values

```
#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;
}
```

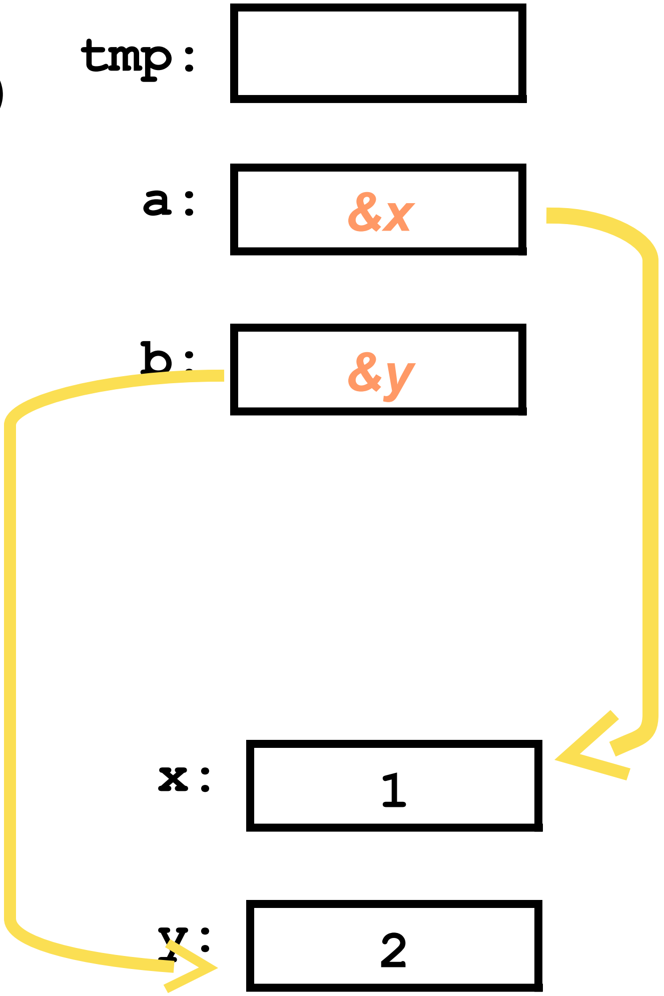
tmp:	1
a:	2
b:	1



x:	1
y:	2

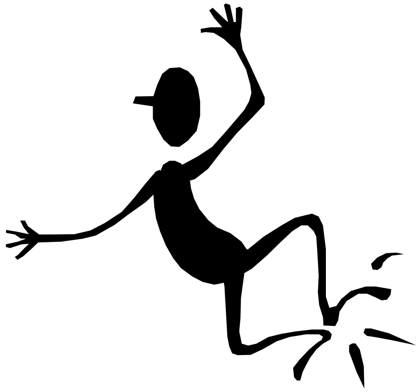
Passing by reference

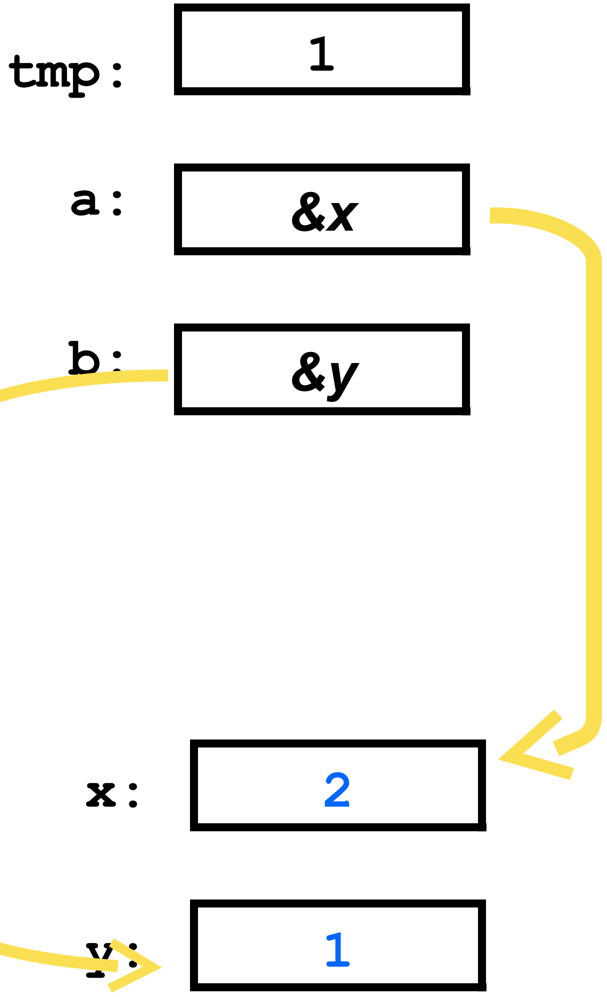
```c
#include <stdio.h>

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;
}
```



Passing by reference

```
#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 as function’s parameters

- Allow to change the value of actual variables.

- Passing by reference in the scanf function

```
char ch;
int  numx;
float numy;
scanf(“%c %d %f”, &ch, &numx, &numy);
```

Advantages

- Passing by references is more effective than passing by values since it does not create a copy of the input values for the function each time it is called.
- Can use parameters as references for creating a function that returns more than one value.

```
void maxmin(int a, int b,
            int *max, int *min)
{
    *max = (a>b) ? a : b;
    *min = (a<b) ? a : b;
}
```

Disadvantages

- It is difficult to control a program using functions with pointer parameters since a variable can be changed anywhere in the program.
- Only use functions with pointer parameters when necessary

Array and pointer

- In an array declaration, the array's name initiates the address where the array is allocated.
- An array corresponds to the address of its first element.

Example:

```
int A[10];
int *ptr;
ptr = A; /* ptr = &A[0] */
```

- ptr[i] and A[i] have the same meaning because ptr and A point to the same address.
- Cannot change the address of an array but can change the address of a pointer

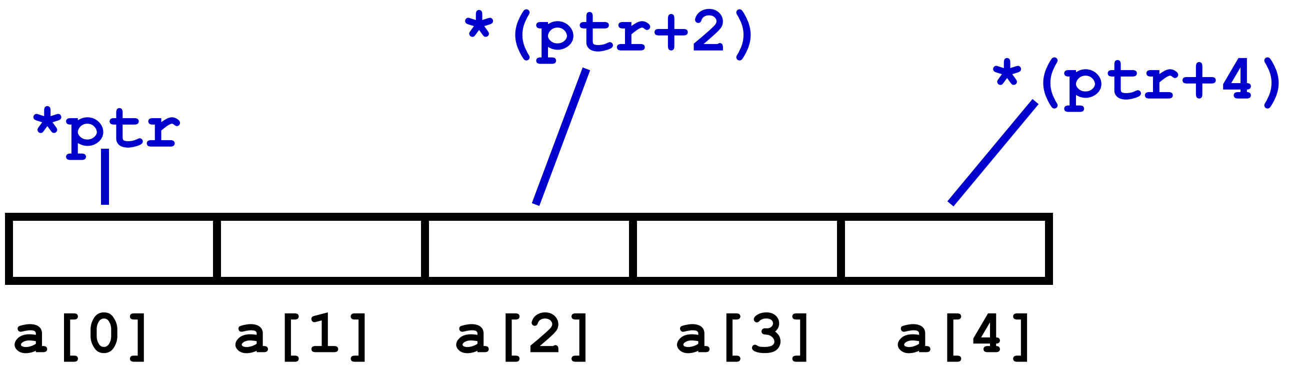
Example:

```
int B[10];
ptr = B; /* OK */
A = B;   /* Not OK */
```

Array and pointer

- Integer math operations can be used with pointers.
- If you increase a pointer, it will be increased by the size of whatever it points to.
- p++ and p += 1 have the same meaning

```
int *ptr = a;
```



```
int a[5];
```

Exercise

- Write a program that includes the following functions:
  - Input values for an array
  - Increase all values of the array by 2
  - Print out the new array
- You should use pointers to access the array. The array is passed as function parameter

Exercise

Are each of the following definitions valid or invalid? If any are invalid, why?

a. int ivar;

   int *iptr = &ivar;

b. int ivar, *iptr = &ivar;

c. float fvar;

   int *iptr = &fvar;

d. int nums[50], *iptr = nums;

e. int *iptr = &ivar;

   int ivar;

Exercise

1. Declare an array of type unsigned int called values with five elements, and initialize the elements to the even integers from 2 to 10. Assume that the symbolic constant SIZE has been defined as 5.
2. Declare a pointer vPtr that points to an object of type unsigned int.
3. Use a for statement to print the elements of array values using array subscript notation.
4. Use a for statement to print the elements of array values using pointer/offset notation.
5. Use a for statement to print the elements of array values by subscripting the pointer to the array.

Exercise

6. Refer to the fifth element of values using array subscript notation, pointer/offset notation with the array name as the pointer, pointer subscript notation and pointer/offset notation.

Assume that unsigned integers are stored in two bytes and that the starting address of the array is at location 1002500 in memory.

6. What address is referenced by vPtr + 3? What value is stored at that location?
7. Assuming that vPtr points to values[4], what address is referenced by vPtr -= 4? What value is stored at that location?