Mahidol University

ITCS113
Fundamentals of Programming

Lecture 11 - Pointer

Instructor: Asst. Prof. Dr. Akara Supratak

Contact: akara.sup@mahidol.edu

Agenda

- Pointers and Addresses
- Pass by Reference
- Using Pointer with Array

Pointers and Addresses

Concept of Variables

Variable: a symbolic name associated with a value; its value can be varied.

- C compiler assigns a specific block of memory within the computer to hold the value of that variable.
- The size of that block depends on the data type.

```
datatype variable_name;
```

Concept of Variables

There are **two** components associated with each variable.

- Value
- Memory address

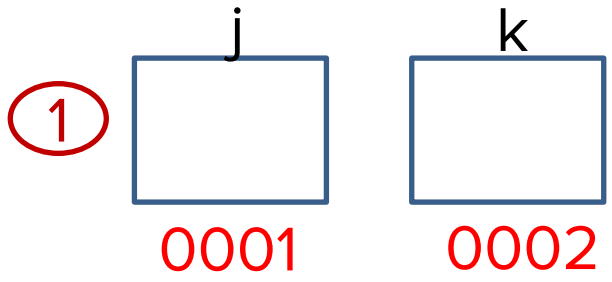
```
int j, k;
k = 2;
j = 7;
k = j;
```

Concept of Variables

There are **two** components associated with each variable.

- Value
- Memory address

```
int j, k;
k = 2;
j = 7;
k = j;
```



Concept of Variables

There are **two** components associated with each variable.

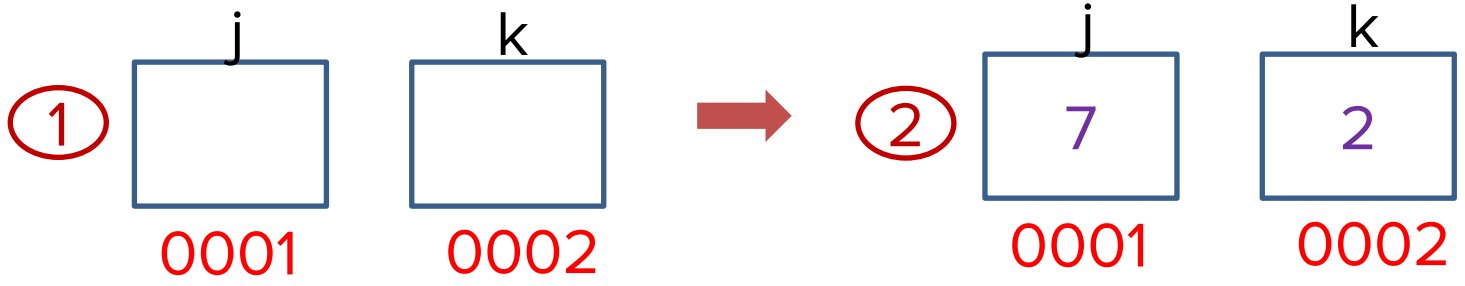
- Value
- Memory address

① `int j, k;`

② `k = 2;`

② `j = 7;`

`k = j;`

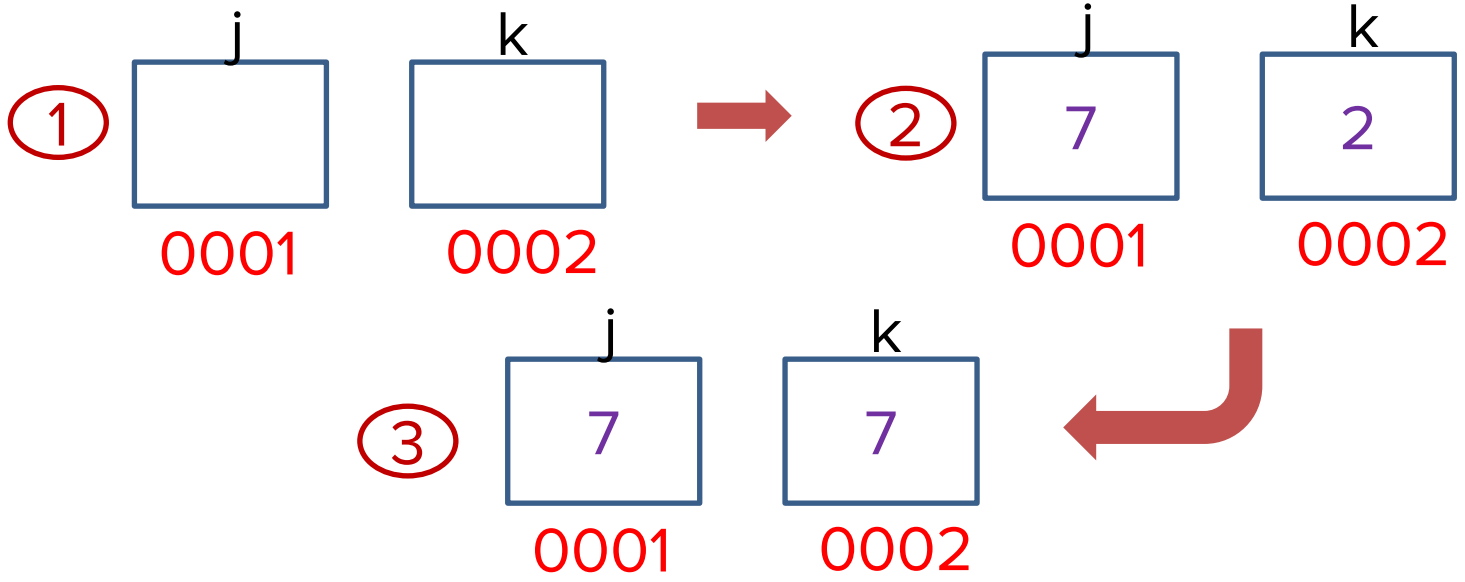


Concept of Variables

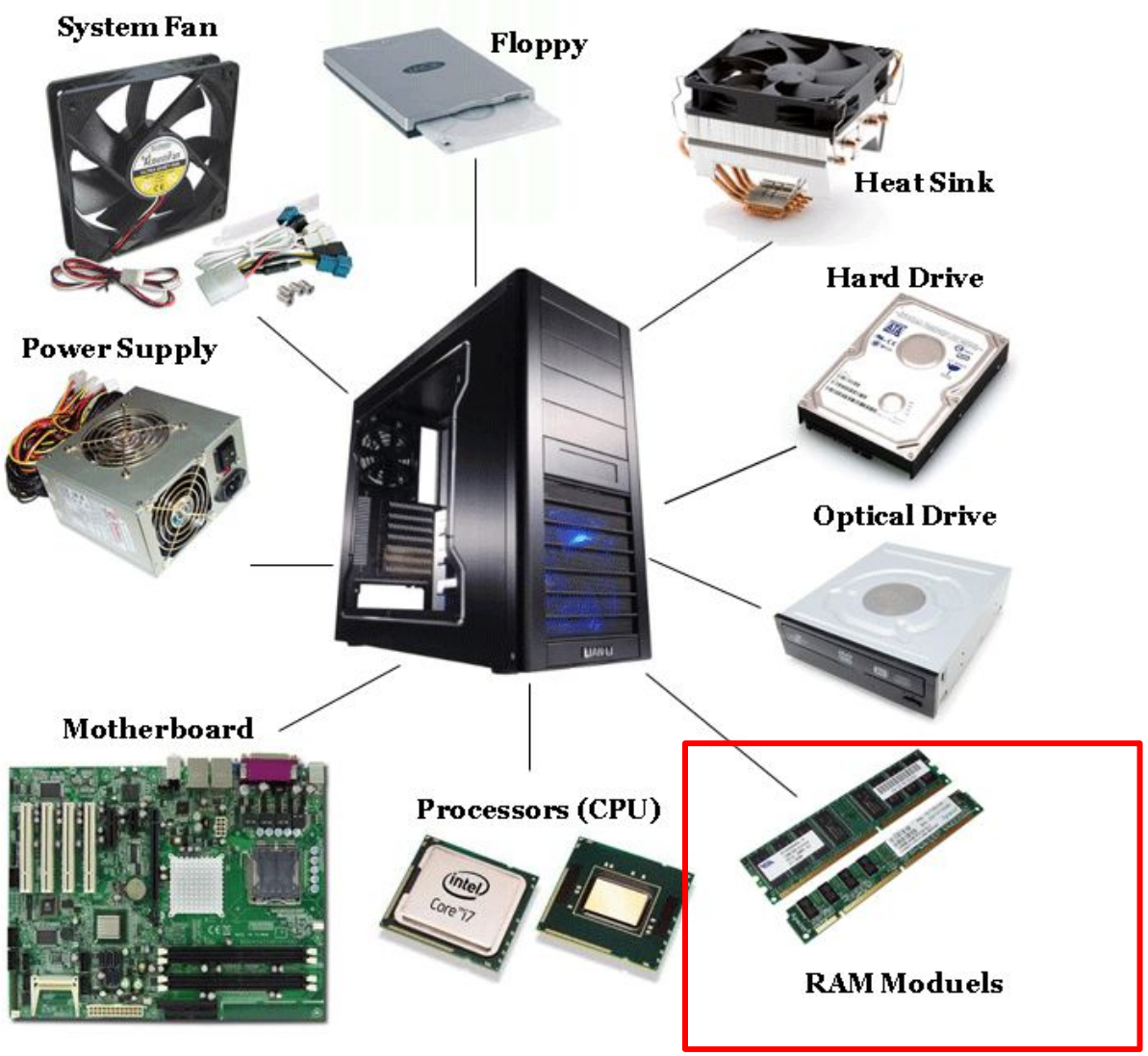
There are **two** components associated with each variable.

- Value
- Memory address

```
① int j, k;
  k = 2;
② j = 7;
③ k = j;
```

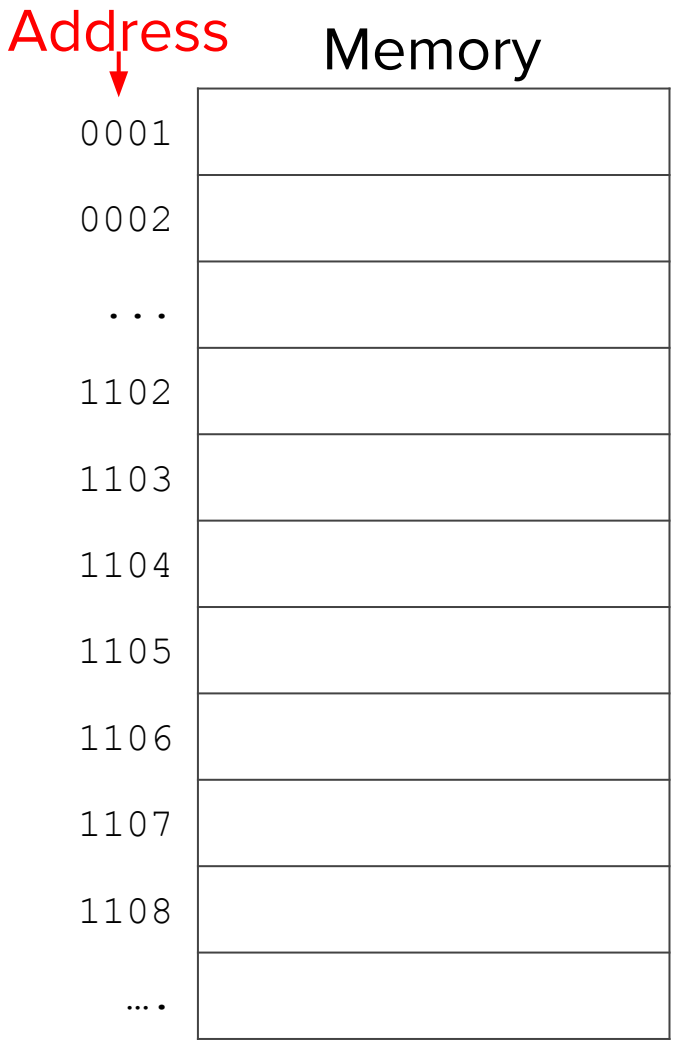


Look Inside Computer Memory

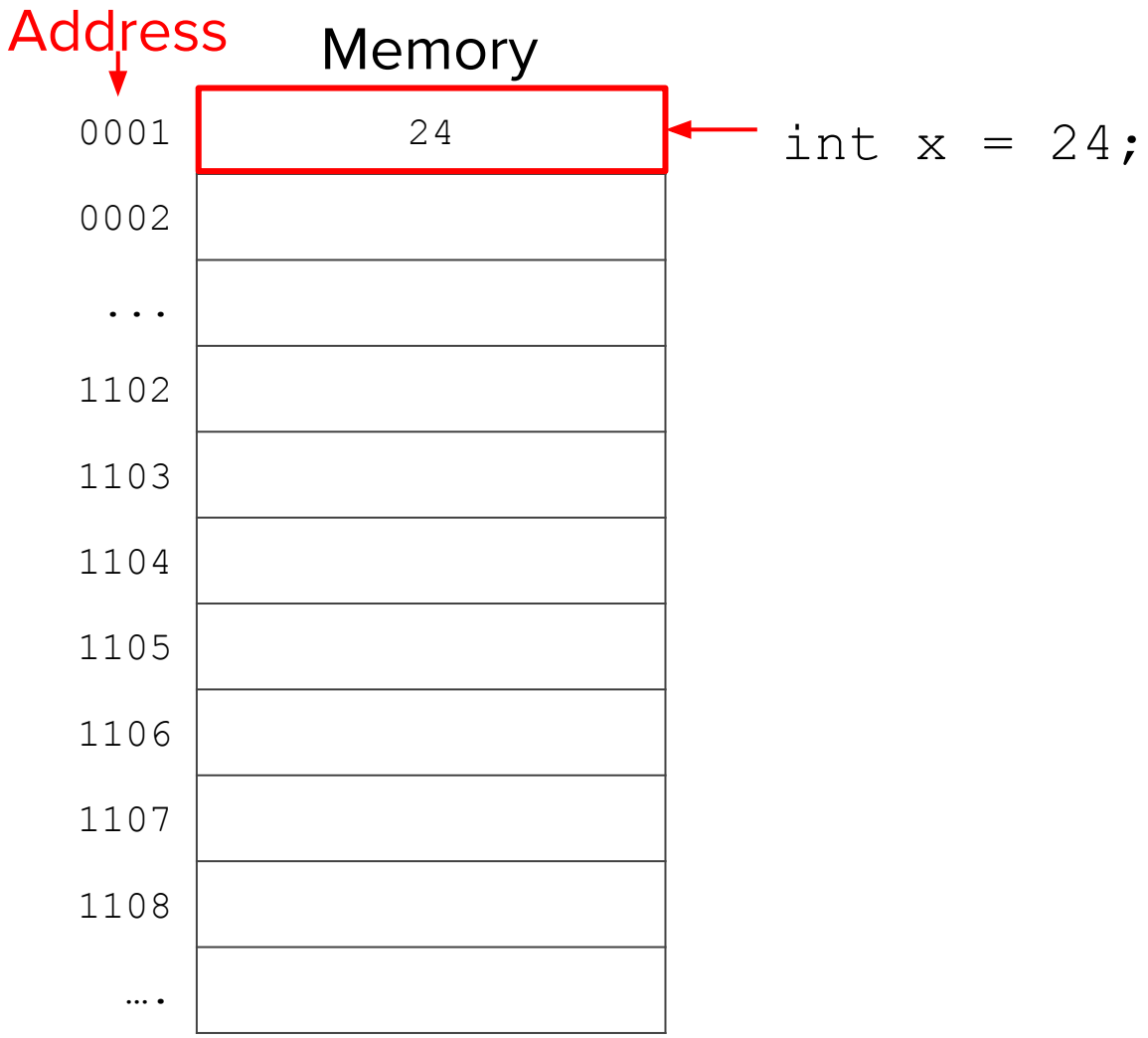


Ref: https://magelmshosting.com/default/component-of-a-computer-system.html

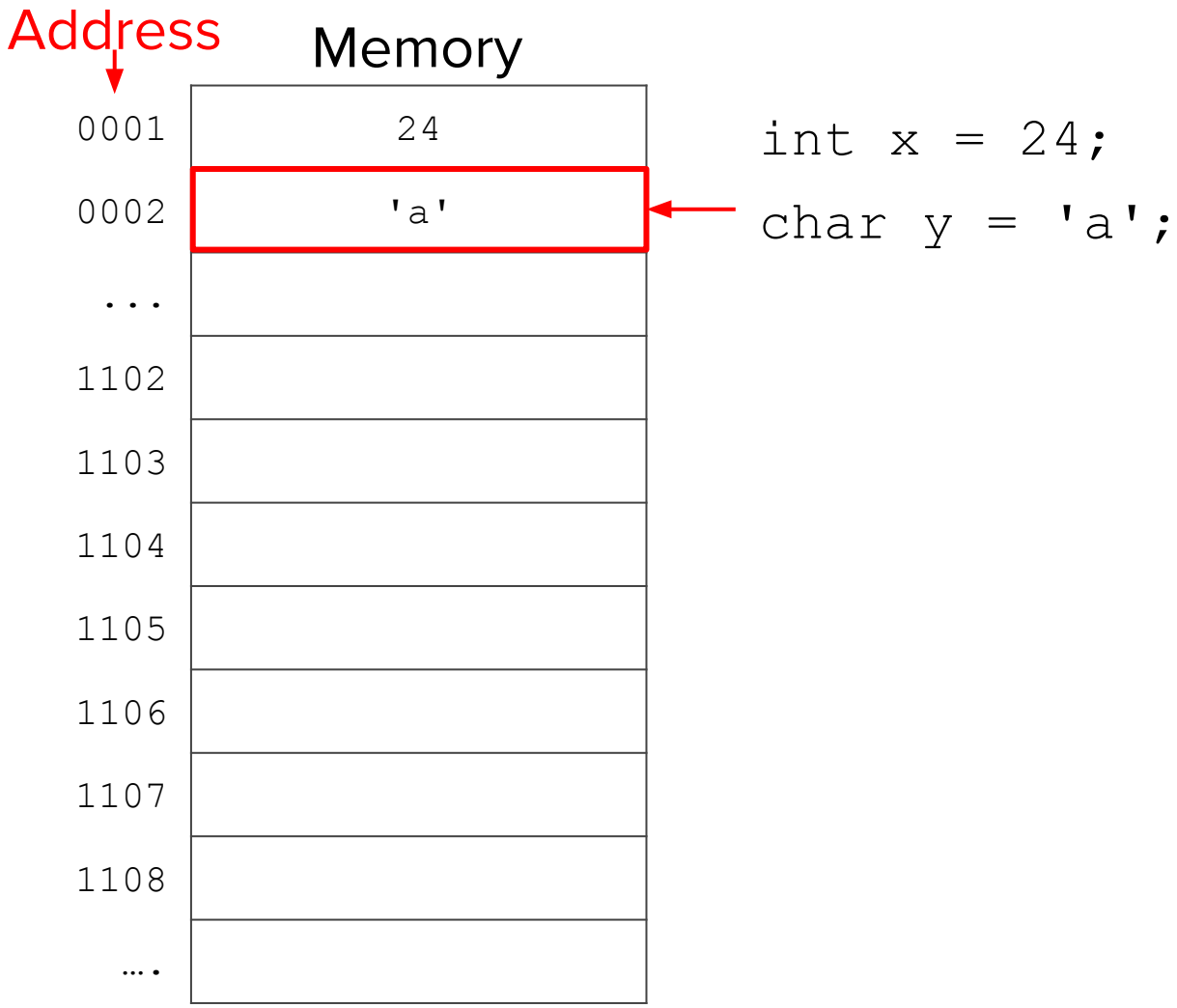
Look Inside Computer Memory



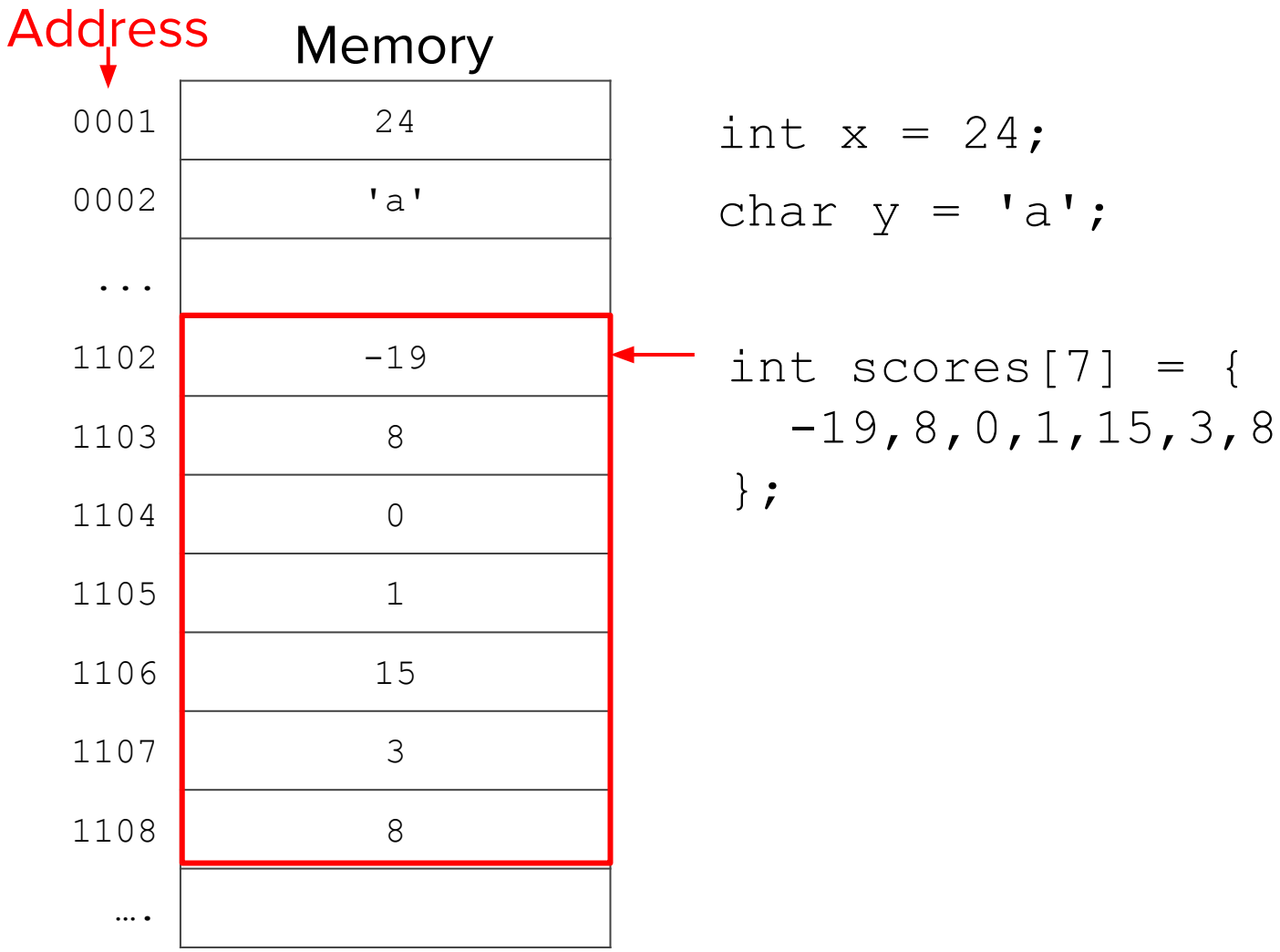
Look Inside Computer Memory



Look Inside Computer Memory

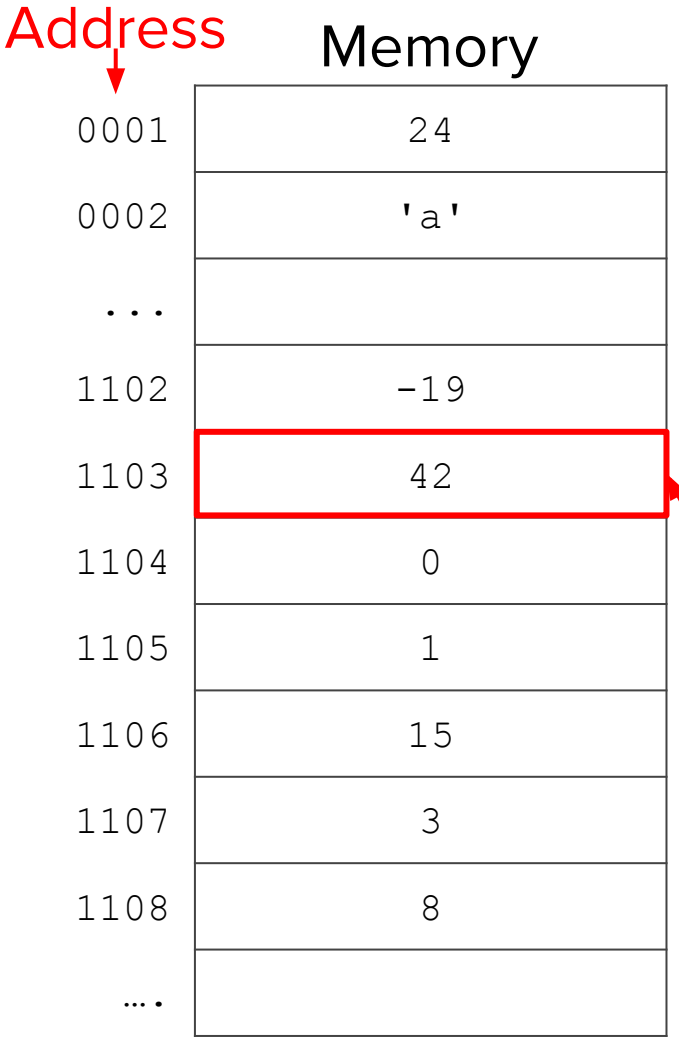


Look Inside Computer Memory




```
int x = 24;
char y = 'a';

int scores[7] = {
  -19,8,0,1,15,3,8
};
```

Look Inside Computer Memory

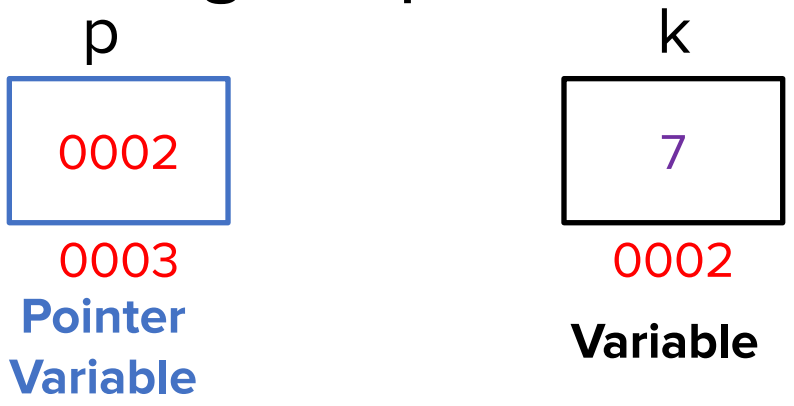



```
int x = 24;
char y = 'a';

int scores[7] = {
  -19,8,0,1,15,3,8
};

scores[1] = 42;
```

What is Pointer?

Pointer: a variable that stores the memory address of another variable located in computer memory.

- A pointer ***references*** a location in memory (e.g., 0002)
- Obtaining the value (e.g., 7) stored at that location (e.g., 0002) is known as ***dereferencing*** the pointer.



Why Pointer?

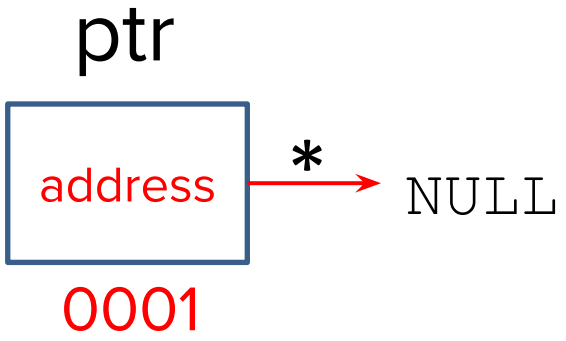
- Modify the values of the variables of the function’s caller.
- Pass an (big) array to a function.
- etc.

Pointer Declaration

```
datatype *variable_name;
```

- `*` (asterisk) is to inform the C compiler that we want a pointer variable.
- `datatype` is used to specify the type of the value that the pointer will point to.
- E.g., a pointer that can point to an integer variable:

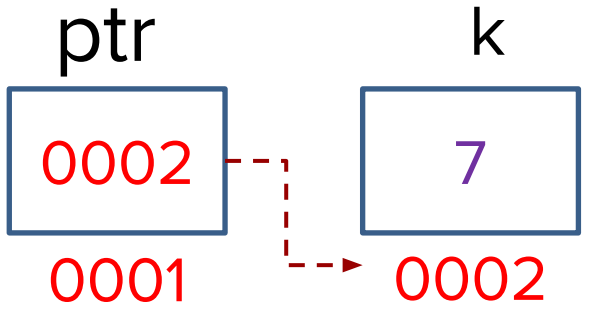
```
int *ptr;
```



Point to a variable

When we want a pointer to point to a variable, we will assign the address of that variable to the pointer.

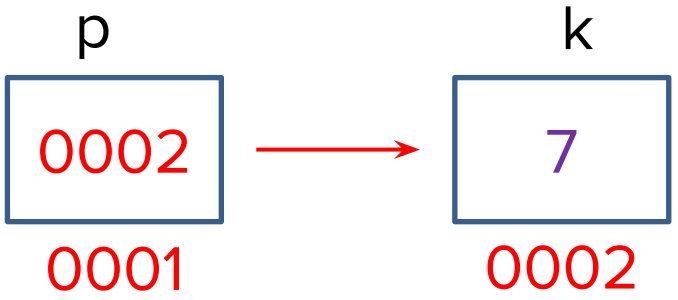
```
ptr = &k;
```



& operator retrieves the address of the variable `k`.

Now, `ptr` is said to "point to k".

Dereferencing



Dereferencing: Obtain the value of the variable that a pointer is pointing to.

- The “dereferencing operator” or “indirection operator” is the asterisk (*****)

```
*ptr
```

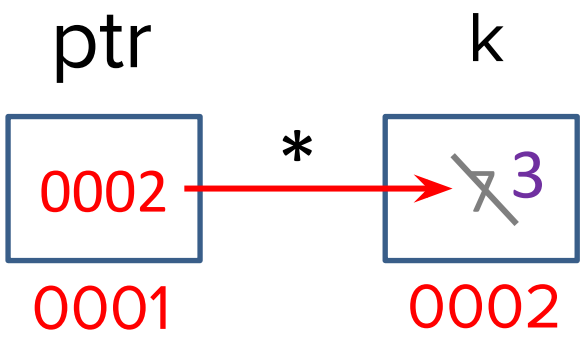
```
printf("%d",*ptr);
int a = *ptr;
int b = a + *ptr;
if (*ptr > 10) {
    …
}
```

```
printf("%d",k);
int a = k;
int b = a + k;
if (k > 10) {
    …
}
```

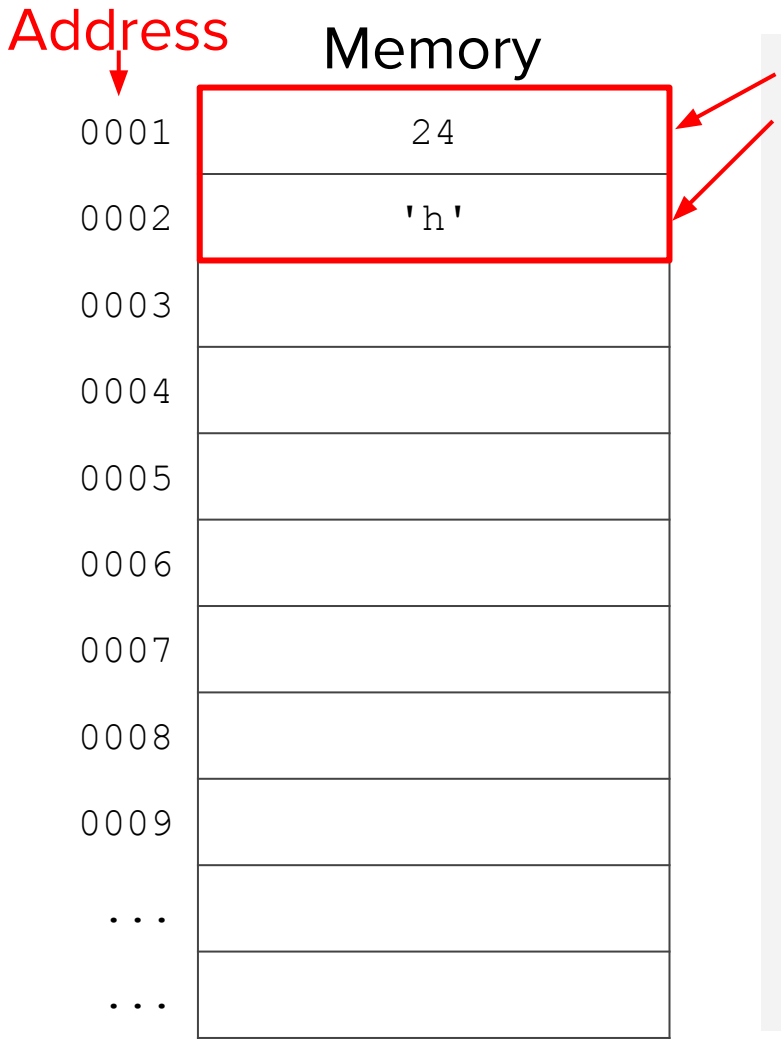
Dereferencing

We can also change the value of the variable that the pointer is pointing to:

```
*ptr = 3;
```

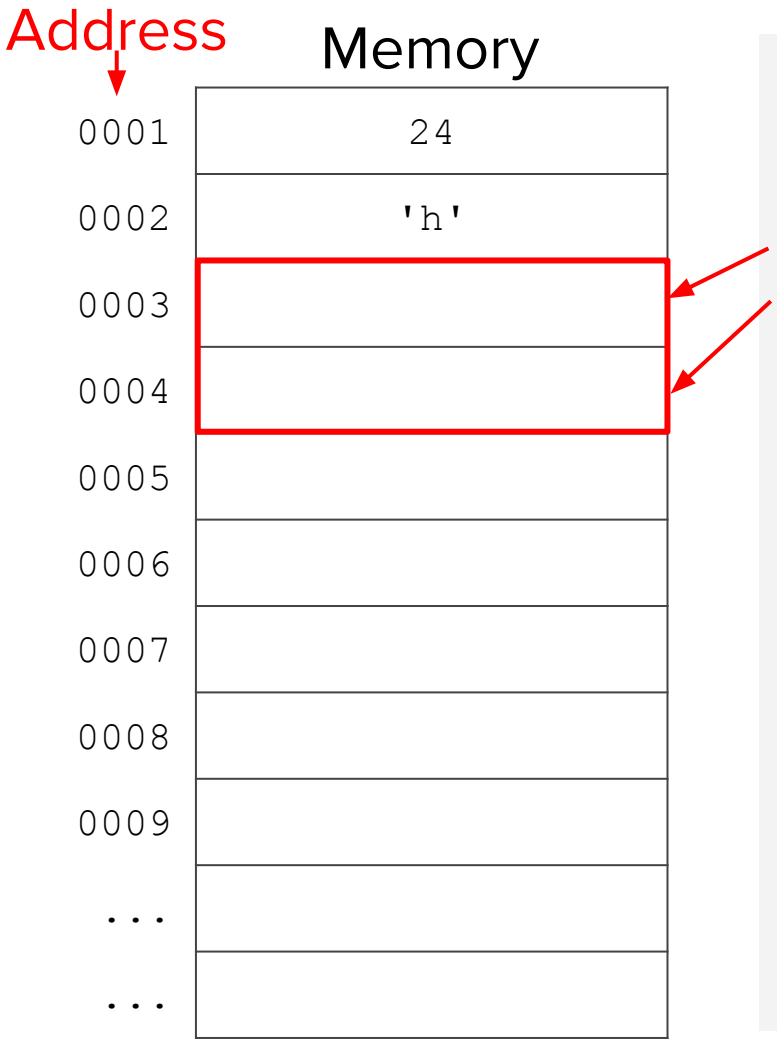


Example



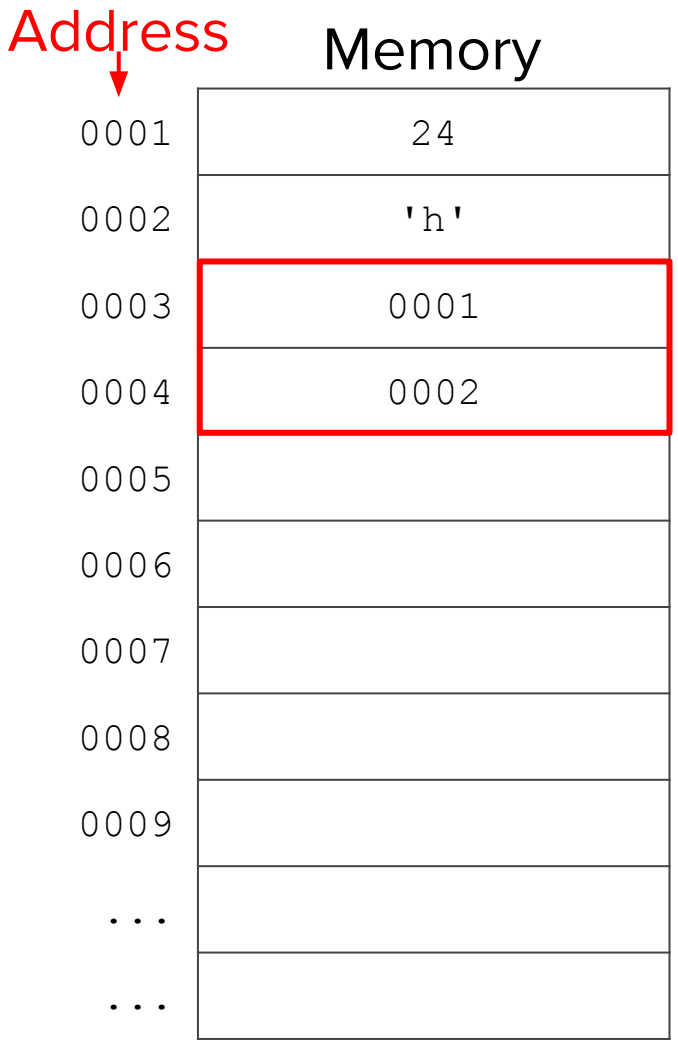

```
int num = 24;       // create an integer
char alpha = 'h';   // create a character
```

Example



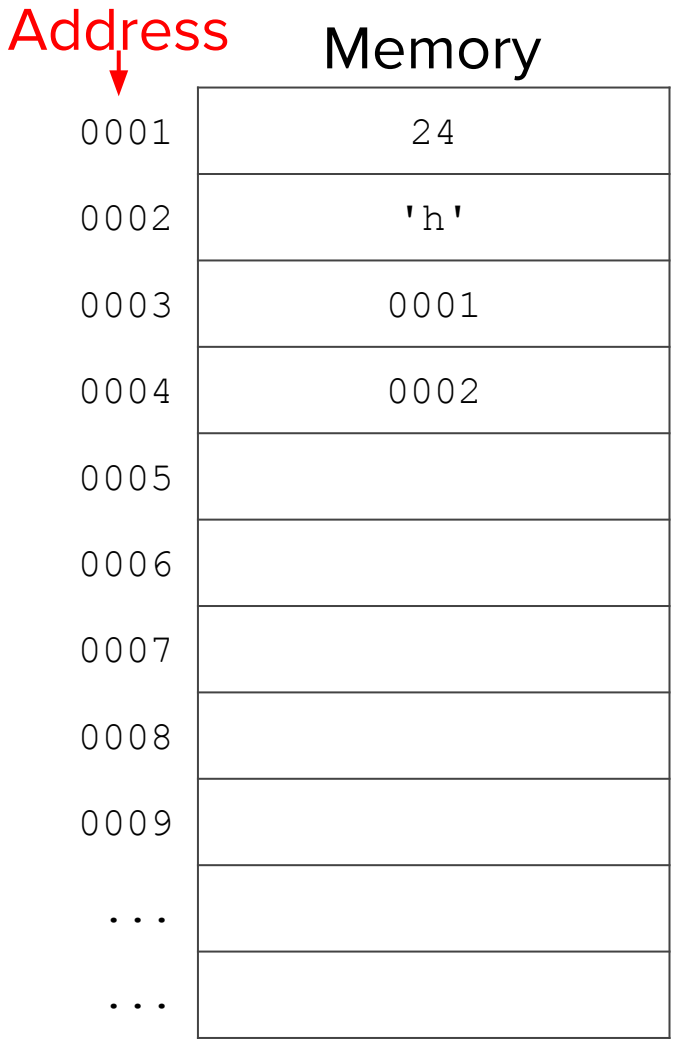

```
int num = 24;       // create an integer
char alpha = 'h';   // create a character

int *p_num;      // create a pointer to an integer
char *p_alpha;   // create a pointer to a character
```

Example



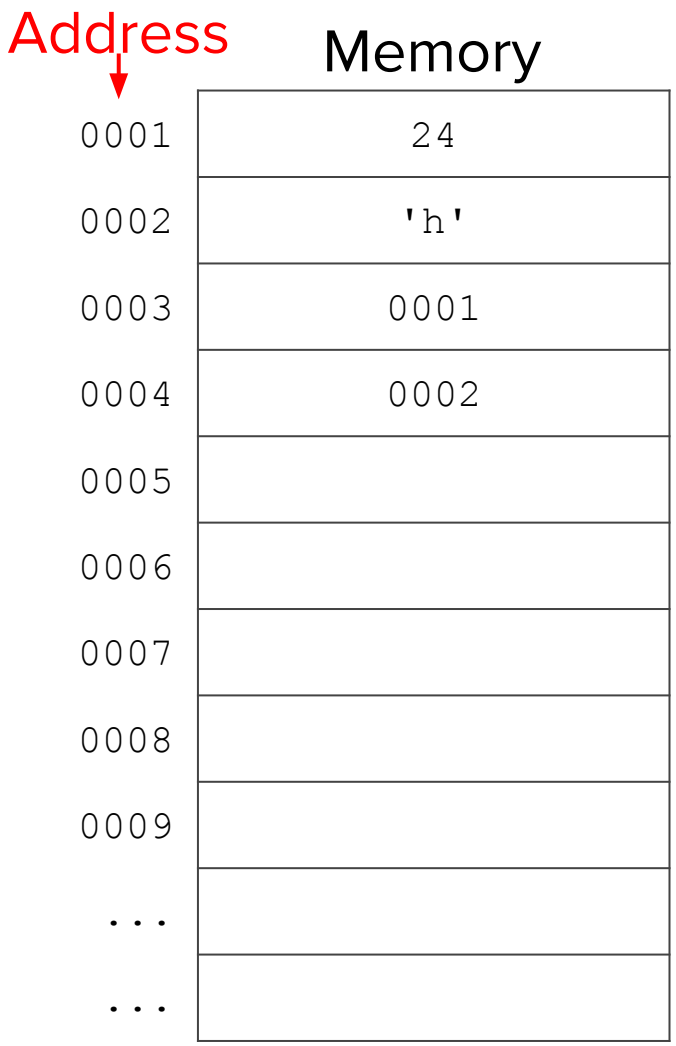

```
int num = 24;       // create an integer
char alpha = 'h';   // create a character

int *p_num;     // create a pointer to an integer
char *p_alpha;  // create a pointer to a character

// Assign the address of num to p_num
p_num = &num;

// Assign the address of alpha to p_alpha
p_alpha = &alpha;
```

Example




```
int num = 24;       // create an integer
char alpha = 'h';   // create a character

int *p_num;     // create a pointer to an integer
char *p_alpha;  // create a pointer to a character

// Assign the address of num to p_num
p_num = &num;

// Assign the address of alpha to p_alpha
p_alpha = &alpha;

printf("%d %d", num, *p_num);       // ? ?
printf("%c %c", alpha, *p_alpha);   // ? ?

printf("Addr of num: %p\n", &num);          // ?
printf("Value of p_num: %p\n", p_num);      // ?
printf("Addr of alpha: %p\n", &alpha);      // ?
printf("Value of p_alpha: %p\n", p_alpha);  // ?
```

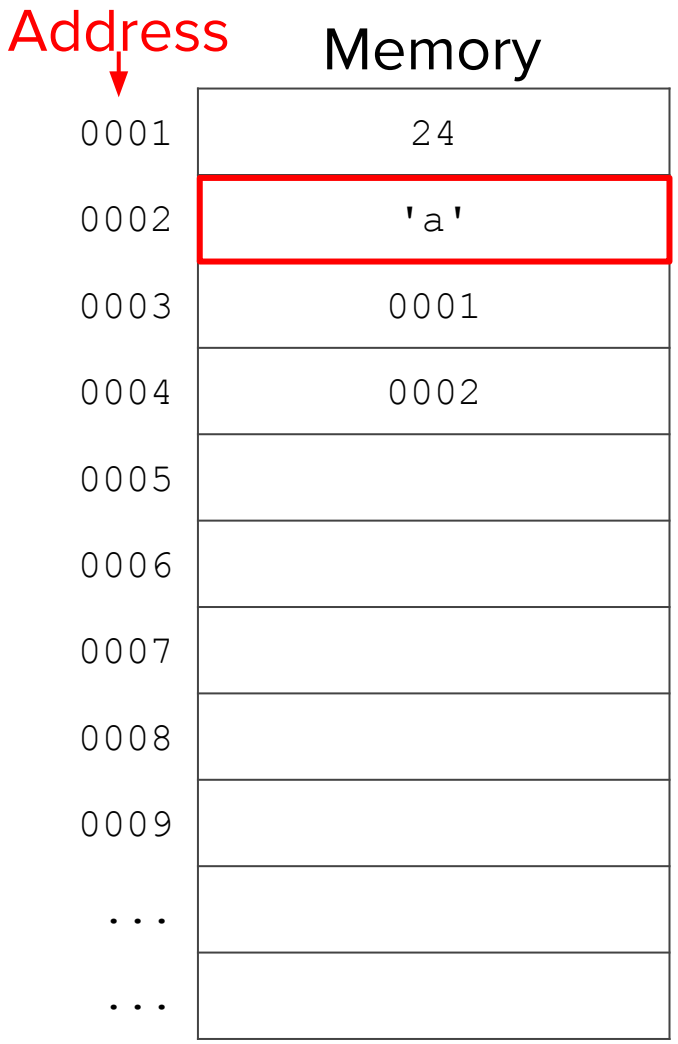
Example




```c
int num = 24;       // create an integer
char alpha = 'h';   // create a character

int *p_num;     // create a pointer to an integer
char *p_alpha;  // create a pointer to a character

// Assign the address of num to p_num
p_num = &num;

// Assign the address of alpha to p_alpha
p_alpha = &alpha;

printf("%d %d", num, *p_num);       // 24 24
printf("%c %c", alpha, *p_alpha);  // h h

printf("Addr of num: %p\n", &num);          // 0x7ffcffbeeba4
printf("Value of p_num: %p\n", p_num);      // 0x7ffcffbeeba4
printf("Addr of alpha: %p\n", &alpha);      // 0x7ffcffbeeba3
printf("Value of p_alpha: %p\n", p_alpha); // 0x7ffcffbeeba3
```

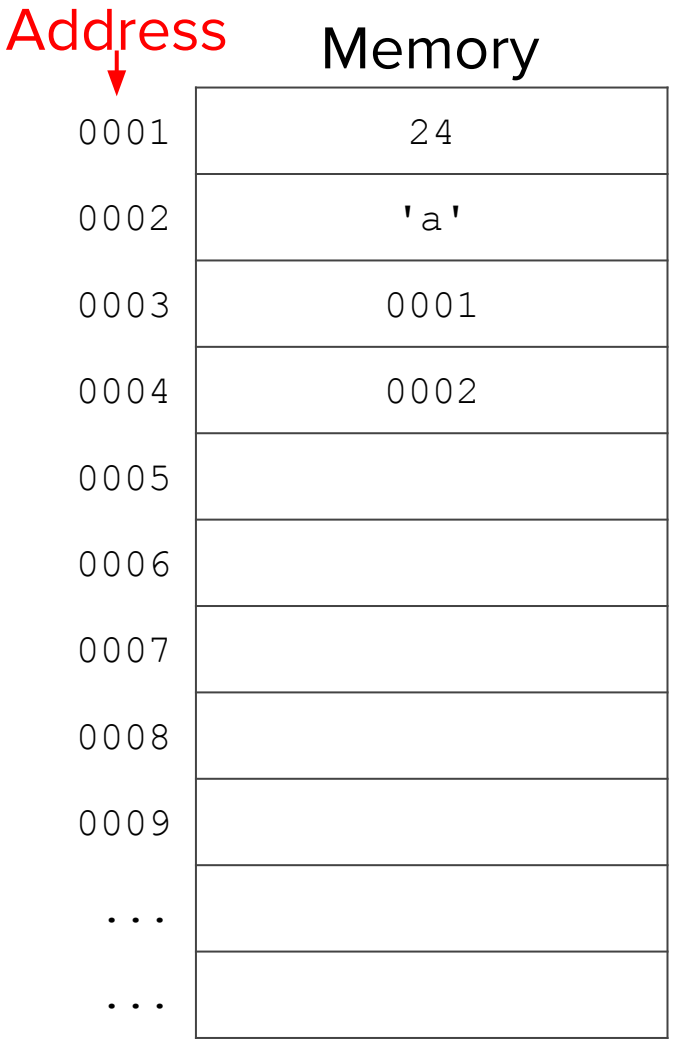
It depends on your computer during runtime

Example



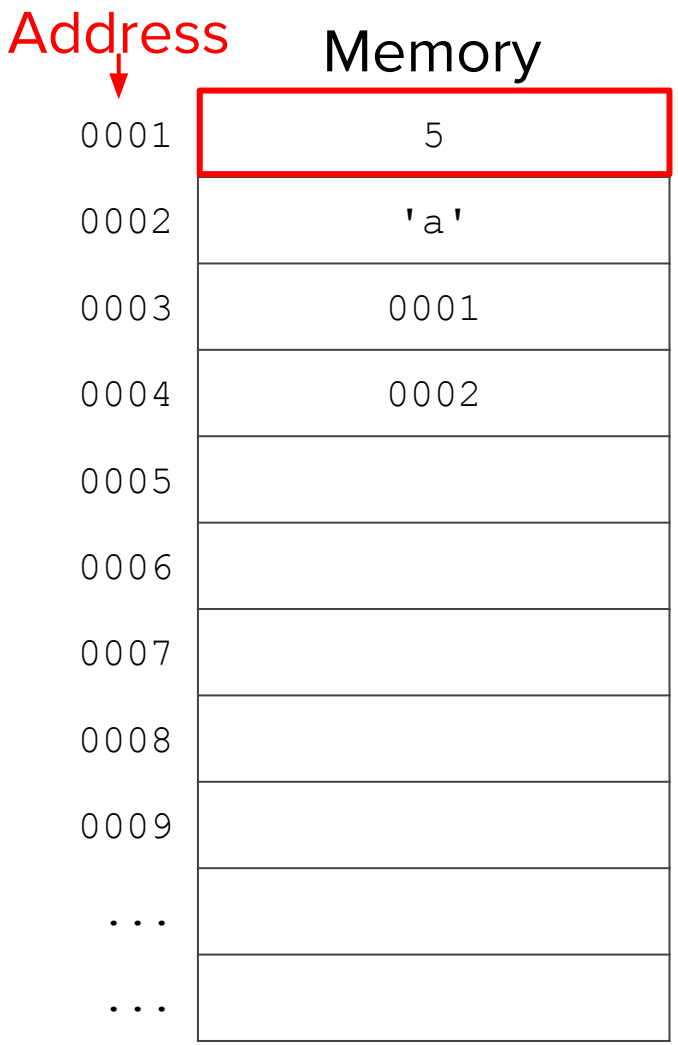

```c
int num = 24;       // create an integer
char alpha = 'h';   // create a character

int *p_num;     // create a pointer to an integer
char *p_alpha;  // create a pointer to a character

// Assign the address of num to p_num
p_num = &num;

// Assign the address of alpha to p_alpha
p_alpha = &alpha;

printf("%d %d", num, *p_num);
printf("%c %c", alpha, *p_alpha);

// Change value of `alpha`
alpha = 'a';

printf("%c %c", alpha, *p_alpha);  // ??
```

Example



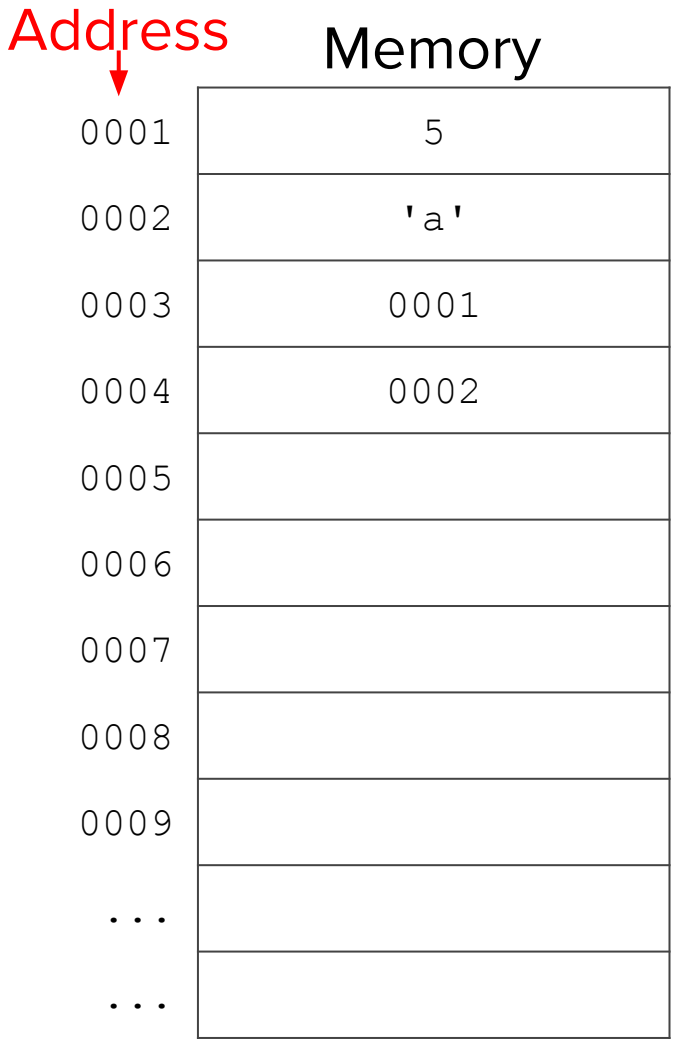

```c
int num = 24;       // create an integer
char alpha = 'h';   // create a character

int *p_num;     // create a pointer to an integer
char *p_alpha;  // create a pointer to a character

// Assign the address of num to p_num
p_num = &num;

// Assign the address of alpha to p_alpha
p_alpha = &alpha;

printf("%d %d", num, *p_num);
printf("%c %c", alpha, *p_alpha);

// Change value of `alpha`
alpha = 'a';

printf("%c %c", alpha, *p_alpha);  // a a
```

Example




```c
int num = 24;       // create an integer
char alpha = 'h';   // create a character

int *p_num;      // create a pointer to an integer
char *p_alpha;   // create a pointer to a character

// Assign the address of num to p_num
p_num = &num;

// Assign the address of alpha to p_alpha
p_alpha = &alpha;

printf("%d %d", num, *p_num);
printf("%c %c", alpha, *p_alpha);

// Change value of `alpha`
alpha = 'a';

printf("%c %c", alpha, *p_alpha);

// Change value at the address from `p_num`
// (i.e., dereferencing)
*p_num = 5;

printf("%d %d", num, *p_num);  // ? ?
```

Example




```c
int num = 24;       // create an integer
char alpha = 'h';   // create a character

int *p_num;      // create a pointer to an integer
char *p_alpha;   // create a pointer to a character

// Assign the address of num to p_num
p_num = &num;

// Assign the address of alpha to p_alpha
p_alpha = &alpha;

printf("%d %d", num, *p_num);
printf("%c %c", alpha, *p_alpha);

// Change value of `alpha`
alpha = 'a';

printf("%c %c", alpha, *p_alpha);

// Change value at the address from `p_num`
// (i.e., dereferencing)
*p_num = 5;

printf("%d %d", num, *p_num);  // 5 5
```

Exercise

```
int num = 17;
int *p_num;
p_num = &num;

printf("%d", *p_num);   // Output?

num = 14;
int x = *p_num;
*p_num = -7;

printf("%d", x);   // Output?

printf("%d", num);   // Output?
```

Summary

Pointer Declaration

Point to a variable

Dereferencing

```
int *p_num;

p_num = &x;

a = 15 + *p_num;
```

	Variable (int x)	Pointer (int *ptr)
Value	x	*ptr
Address	&x	ptr

Pass by Reference

Function Call

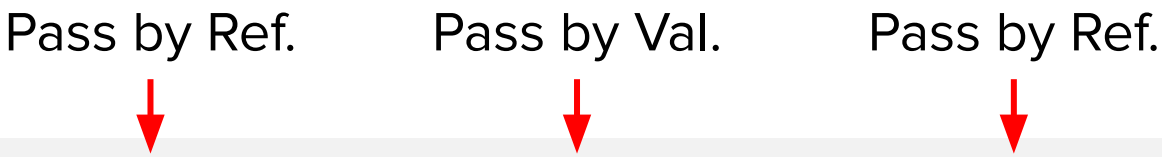
Pass by value: Passing copies of values of variables to a function

Pass by reference: Passing copies of addresses of variables to a function

Function Call

Pass by value: Passing copies of values of variables to a function

Pass by reference: Passing copies of addresses of variables to a function




```
return_type function_name(type1 *name1, type2 name2, type3 *name3)
{
    statement1;  // may define new params
    statement2;  // may use arguments
    …
    return expression;
}
```

Function Call

Pass by value: Passing copies of values of variables to a function

Pass by reference: Passing copies of addresses of variables to a function

Pass by Ref. Pass by Val. Pass by Ref.

```
return_type function_name(type1 *name1, type2 name2, type3 *name3)
{
    statement1;  // may define new params
    statement2;  // may use arguments
    …
    return expression;
}
```

Inside the function, they can be used in **exactly the same way as the pointers**.

Why Pass by Reference?

- A function can change the value of the arguments
- A function can receive and process arrays (discussed later)

Example

```c
int main() {
    int num1 = 5;
    float num2 = -4.78;
    char char1 = 'a';
    float *p_float;
    char *p_char;
    p_float = &num2;  // point to num2
    p_char = &char1;  // point to char1
    func1(num1, p_char, p_float);
    printf("%d %c %.2f\n", num1, char1, num2);
    return 0;
}
```

```c
void func1(int a, char *b, float *c)
{
    printf("%d %c %.2f\n", a, *b, *c);
    a = 60;
    *b = 'm';
    *c = 9.6;
}
```

Example

```c
int main() {
    int num1 = 5;
    float num2 = -4.78;
    char char1 = 'a';
    float *p_float;
    char *p_char;
    p_float = &num2;  // point to num2
    p_char = &char1;  // point to char1
    func1(num1, p_char, p_float);
    printf("%d %c %.2f\n", num1, char1, num2);
    return 0;
}
```

```c
void func1(int a, char *b, float *c)
{
    printf("%d %c %.2f\n", a, *b, *c);
    a = 60;
    *b = 'm';
    *c = 9.6;
}
```

Output:

```
5 a -4.78
5 m 9.60
```

Example

We can also just pass the address of the variables as the input arguments.

```c
int main() {
    int num1 = 5;
    float num2 = -4.78;
    char char1 = 'a';
    func1(num1, &char1, &num2);
    printf("%d %c %.2f\n", num1, char1, num2);
    return 0;
}
```

```c
void func1(int a, char *b, float *c)
{
    printf("%d %c %.2f\n", a, *b, *c);
    a = 60;
    *b = 'm';
    *c = 9.6;
}
```

Output:

```
5 a -4.78
5 m 9.60
```

Example: Swap values

```
void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}

int main()
{
    int x, y;
    x = 5;
    y = 10;
    swap(x, y);
    printf("x=%d y=%d\n", x, y);
    return 0;
}
```

Example: Swap values

```
void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}

int main()
{
    int x, y;
    x = 5;
    y = 10;
    swap(x, y);
    printf("x=%d y=%d\n", x, y);
    return 0;
}
```

DOES NOT exchange the values !!!

Output:

```
x=5 y=10
```

Example: Swap values

```
void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

int main()
{
    int x, y;
    x = 5;
    y = 10;
    swap(&x, &y);
    printf("x=%d y=%d\n", x, y);
    return 0;
}
```

Output:

```
x=10 y=5
```

Exercise

```c
#include <stdio.h>

int func1(int *num1, int num2, int *num3) {
    *num1 += 1; // a = 2
    num2 += 2; // 12
    *num3 += num2 + *num1 + 1; // 115
    printf("%d %d %d\n", *num1, num2, *num3);
    return *num1 + num2 + *num3;
}
```

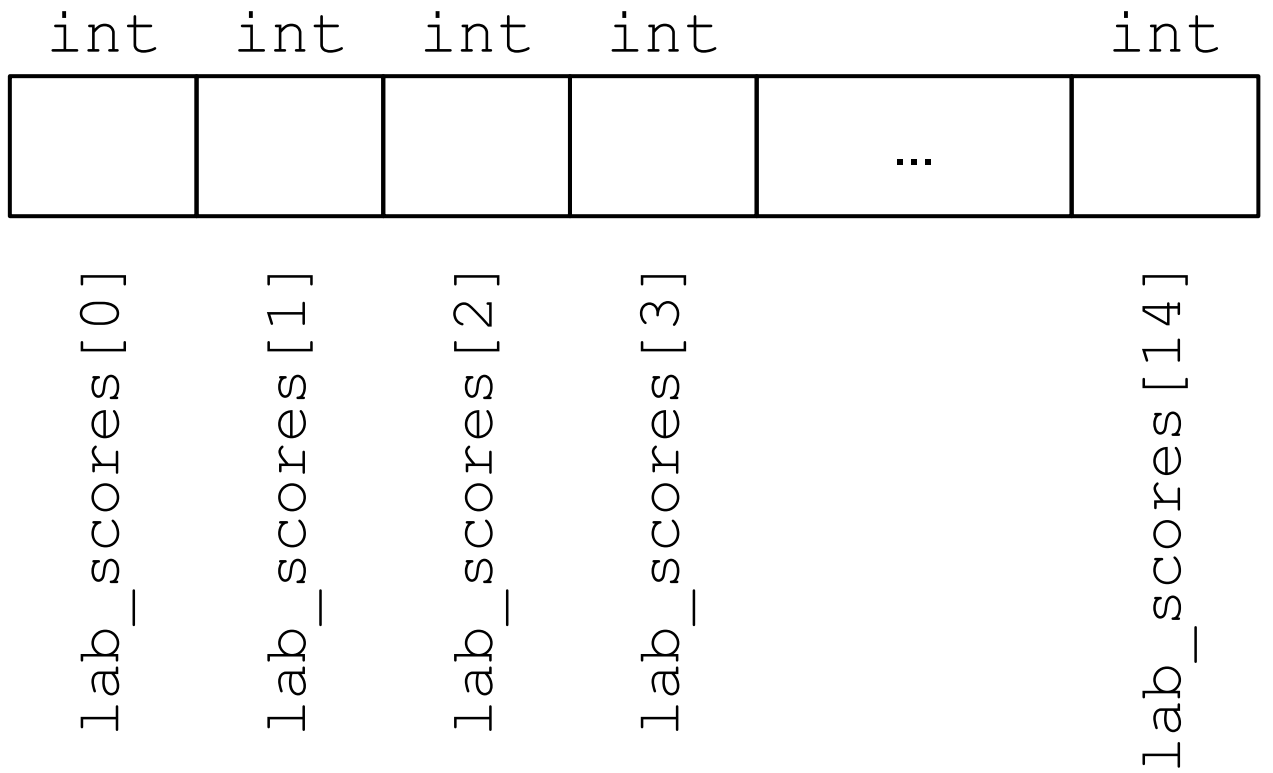
```c
int main()
{
    int a = 1, b = 10, c = 100;

    int *ptr;
    ptr = &c;
    // int *ptr = &c;

    int result = func1(&a, b, ptr);
    printf("%d %d %d\n", a, b, c);
    printf("%d\n", result);

    return 0;
}
```

Using Pointer with Array

Recap: Array

Array: a list of values of the same data type that is stored using a single group name.

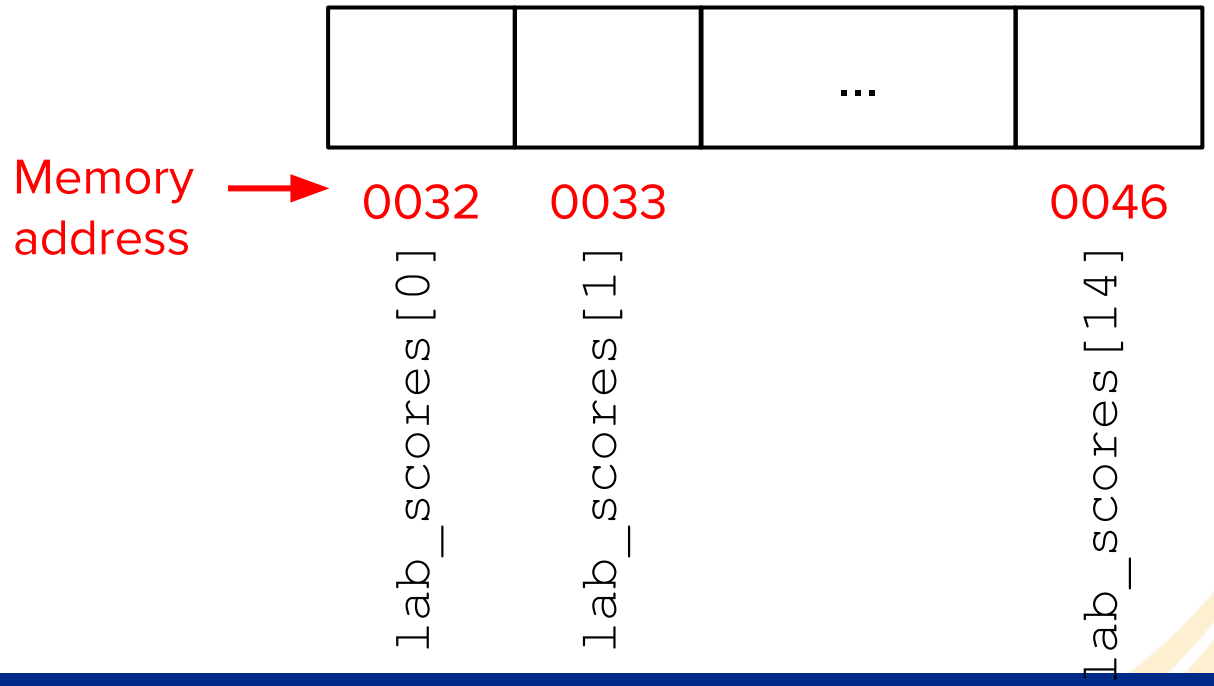
```
int lab_scores[15];
```



Access Array Elements with Pointer

By making a pointer points to the first element of the array, we can use **“pointer + offset”** to access each element in the array.

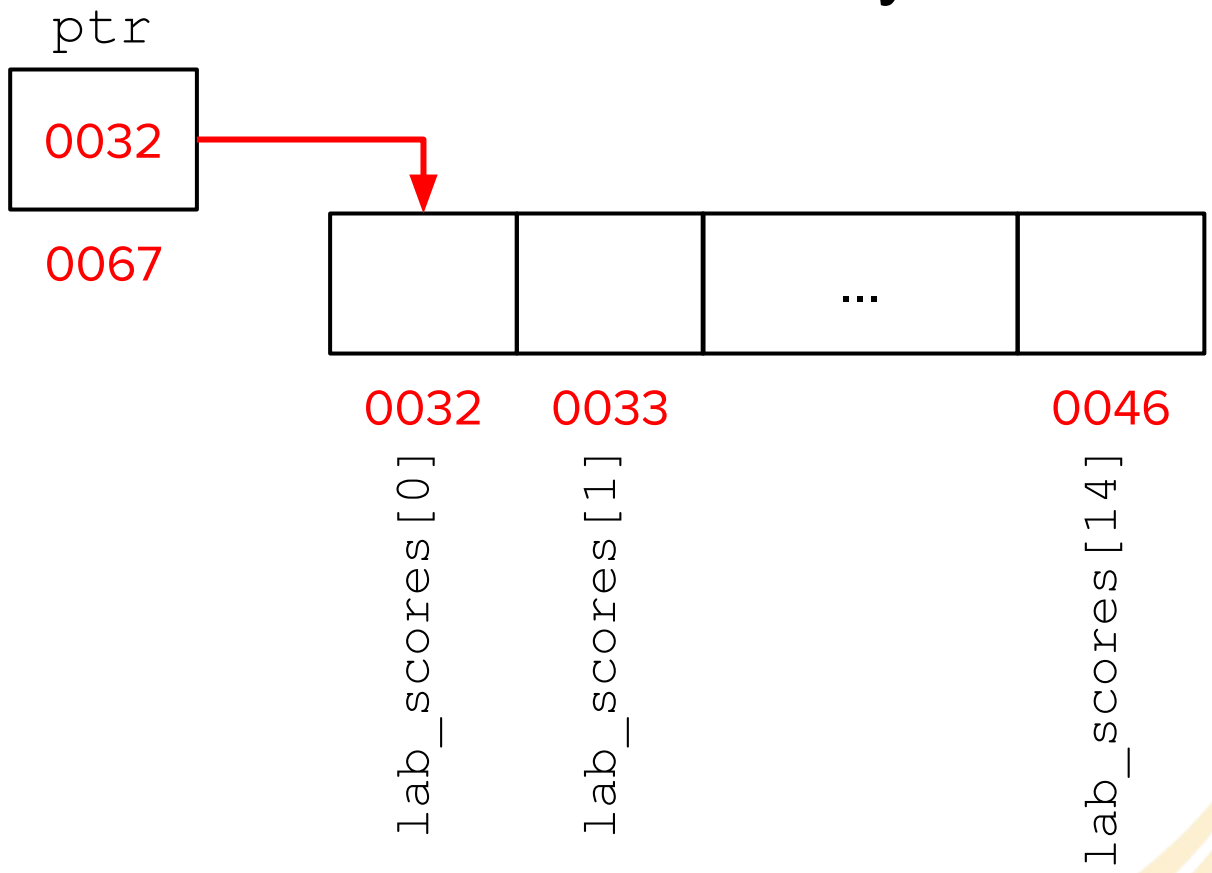
```
int lab_scores[15] = {...};
```



Access Array Elements with Pointer

By making a pointer points to the first element of the array, we can use **“pointer + offset”** to access each element in the array.

```
int lab_scores[15] = {...};

// Point to the 1st element
int *ptr;
ptr = &lab_scores[0];
```

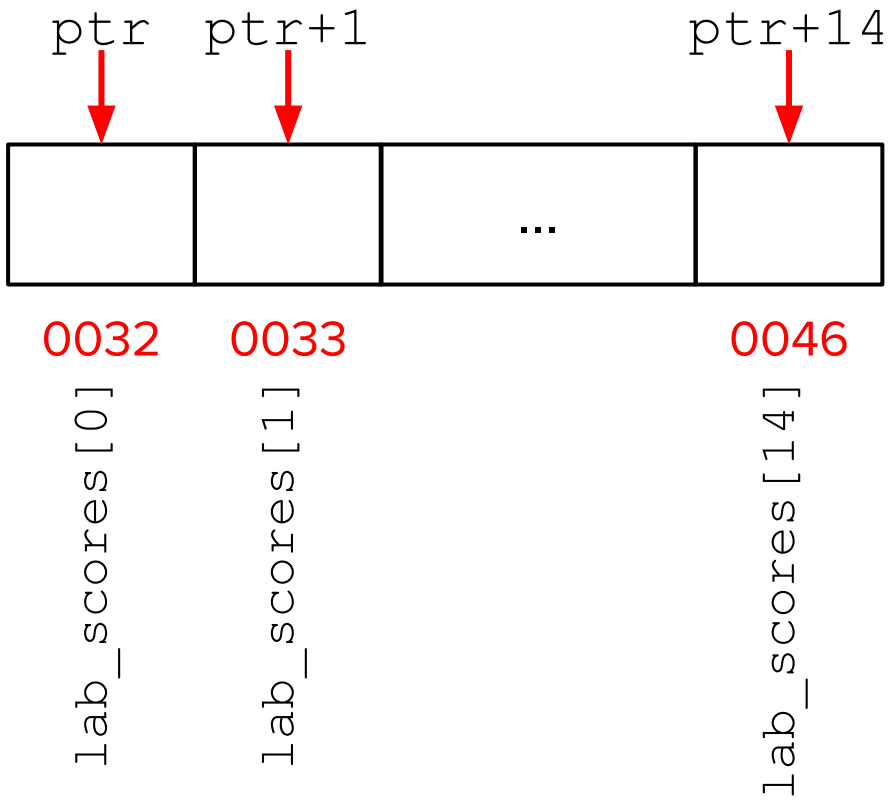


Access Array Elements with Pointer

By making a pointer points to the first element of the array, we can use **“pointer + offset”** to access each element in the array.

```
int lab_scores[15] = {...};

// Point to the 1st element
int *ptr;
ptr = &lab_scores[0];

// Access array elements
int i;
for (i=0 ; i<15 ; i++) {
    printf("%d ", *(ptr+i));
}
```

offset

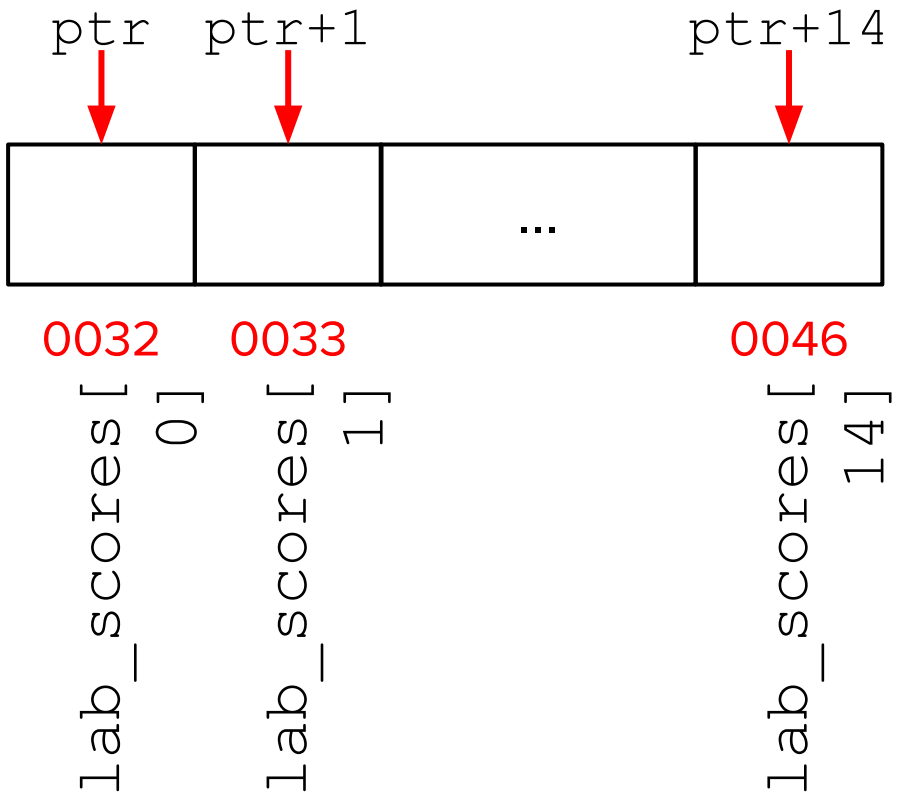


Access Array Elements with Pointer

By making a pointer points to the first element of the array, we can use **“pointer + offset”** to access each element in the array.

```
int lab_scores[15] = {...};

// Point to the 1st element
int *ptr;
ptr = &lab_scores[0];

// Access array elements
int i;
for (i=0 ; i<15 ; i++) {
    printf("%d ", ptr[i]));
}
```

offset

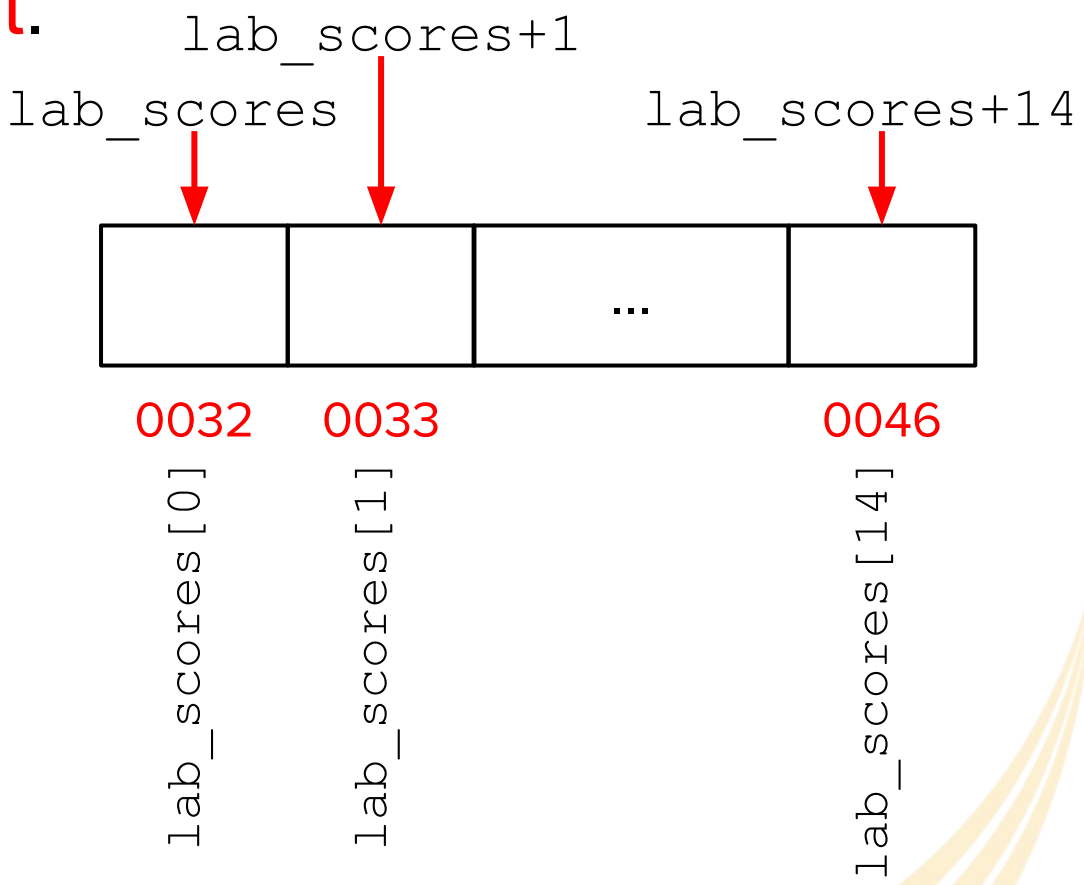


Array Name as Pointer

When we use the array name without the index, the name is converted to a pointer to its first element.

```
int lab_scores[15] = {...};

// Access array elements
int i;
for (i=0 ; i<15 ; i++) {
    printf("%d ", *(lab_scores+i));
}
```

Here, `lab_scores` is converted to a pointer to its first element, which is then added with the offset to access each array element.



Example

```c
int nums[10] = {10,315,72,73,34,25,61,72,18,-9};
int *p_num;
// Point to the 1st element
p_num = &nums[0]; // OR p_num = nums;

for (int i=0 ; i<10 ; i++) {
    printf("%d ", nums[i]);
}
for (int i=0 ; i<10 ; i++) {
    printf("%d ", p_num[i]);
}
for (int i=0 ; i<10 ; i++) {
    printf("%d ", *(nums+i));
}
for (int i=0 ; i<10 ; i++) {
    printf("%d ", *(p_num+i));
}
```

Example

```c
int nums[10] = {10,315,72,73,34,25,61,72,18,-9};
int *p_num;
// Point to the 1st element
p_num = &nums[0]; // OR p_num = nums;

for (int i=0 ; i<10 ; i++) {
    printf("%d ", nums[i]);
}
for (int i=0 ; i<10 ; i++) {
    printf("%d ", p_num[i]);
}
for (int i=0 ; i<10 ; i++) {
    printf("%d ", *(nums+i));
}
for (int i=0 ; i<10 ; i++) {
    printf("%d ", *(p_num+i));
}
```

Output:

```
10 315 72 73 34 25 61 72 18 -9
10 315 72 73 34 25 61 72 18 -9
10 315 72 73 34 25 61 72 18 -9
10 315 72 73 34 25 61 72 18 -9
```

Exercise

```c
#include <stdio.h>
#define N 5


int main()
{
    int nums[N] = {-4, 15, 91, 34, 0};
    int *ptr_1 = &nums[0];
    int *ptr_2 = &nums[4];
    int *ptr_3 = nums;
```

```c
    printf("%d\n", *ptr_1);
    printf("%d\n", *(ptr_1+3));
    printf("%d\n", ptr_1[1]);
    printf("%d\n", nums[1]);
    printf("%d\n", *nums);
    printf("%d\n", *ptr_3);
    printf("%d\n", *ptr_2);
    printf("%d\n", *(ptr_2-2));
    printf("%d\n", *(ptr_2+1));
    return 0;
}
```

Pass Array to Function

We can use **“pass by reference”** to pass an array to a function.

```c
#include <stdio.h>
#define N 5

int find_max(int *arr, int n_elems);

int main()
{
    int nums[N] = {4, -5, 7, 99, 0};
    printf("%d", find_max(&nums[0], N));
}
```

```c
int find_max(int *arr, int n_elems)
{
    int i;
    int max = *arr;
    for (i=1 ; i<n_elems ; i++) {
        if (*(arr+i) > max) {
            max = *(arr+i);
        }
    }
    return max;
}
```

Pass Array to Function

We can use **“pass by reference”** to pass an array to a function.

```c
#include <stdio.h>
#define N 5

int find_max(int *arr, int n_elems);

int main()
{
    int nums[N] = {4, -5, 7, 99, 0};
    printf("%d", find_max(nums, N));
}
```

```c
int find_max(int *arr, int n_elems)
{
    int i;
    int max = *arr;
    for (i=1 ; i<n_elems ; i++) {
        if (*(arr+i) > max) {
            max = *(arr+i);
        }
    }
    return max;
}
```

Pass Array to Function

We can use **“pass by reference”** to pass an array to a function.

```c
#include <stdio.h>
#define N 5

int find_max(int *arr, int n_elems);

int main()
{
    int nums[N] = {4, -5, 7, 99, 0};
    printf("%d", find_max(nums, N));
}
```

```c
int find_max(int *arr, int n_elems)
{
    int i;
    int max = arr[0];
    for (i=1 ; i<n_elems ; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}
```

Pass Array to Function

We can use **“pass by reference”** to pass an array to a function.

```c
#include <stdio.h>
#define N 5

int find_max(int arr[], int n_elems);

int main()
{
    int nums[N] = {4, -5, 7, 99, 0};
    printf("%d", find_max(nums, N));
}
```

```c
int find_max(int arr[], int n_elems)
{
    int i;
    int max = arr[0];
    for (i=1 ; i<n_elems ; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
    }
    return max;
}
```

Exercise

What is the output?

```c
#include <stdio.h>

void func1(int *arr, int n);

int main()
{
    int arr[5] = {-5, 3, 4, 1, 8};
    func1(arr, 5);
    int i;
    for (i=0 ; i<5 ; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

```c
void func1(int *arr, int n)
{
    int i;
    for (i=0 ; i<n ; i++) {
        arr[i] = arr[i] * arr[i];
    }
}
```

Exercise

```c
#include <stdio.h>

// TODO: Function prototype
int main() {
   int n = 5;
   int arr[n];
   int i;
   for (i=0 ; i<n ; i++) {
       arr[i] = i*i;
   }
   // TODO: Call function to compute the sum
   return 0;
}
// TODO: Function definition
```

Lab Exercises