

# Lecture 6

14 September 2020

Admin Matters

Unit 13: Call Stack

Unit 14: Pointers

Unit 15: Arrays

Unit 16: Strings

# Assignment 2

- Due **tomorrow 2359 hrs**
- Graded on **correctness**
  - Syntax
  - Practices
  - Approach
  - Logic
- Also graded on **style** and **efficiency**

# Assignment 2

- Make sure that your submissions compiles cleanly without errors or warnings
  - 0 if cannot compile
  - -1 per warning
- Marks will be deducted if you do not demonstrate a full understanding of what has been taught

# WARNING

- **Plagiarism will not be tolerated**
  - You may discuss how to solve assignment questions
  - But code should be written individually
- **Consequences**
  - 0 marks for the assignment
  - Disciplinary action

# CS1010 Survey

- **Module Evaluation Survey 1**
  - Please help us to improve your learning with constructive feedback!
- Complete by this **Thursday 2359 hrs**

# Upcoming Releases

- Exercise 3: **Tuesday**
- Practical Examination 1 from AY18/19: **Tuesday**
- Assignment 3: **Thursday**
  - Same grading criteria as Assignment 2

# Catch-up II Session

- Postponed
  - New Date: 21 September, Monday
  - Time: 1000 – 1200 hrs
  - Venue: Online – same Zoom link (ref. Piazza)
  - Previously: ~~Saturday, 19 September~~

# Reminders

- Midterm: 28 September 2020, Monday
- Practical Examination 1: 3 October 2020, Saturday
- Important note:
  - Pay attention to examination regulations
  - Disciplinary action will be taken against violators



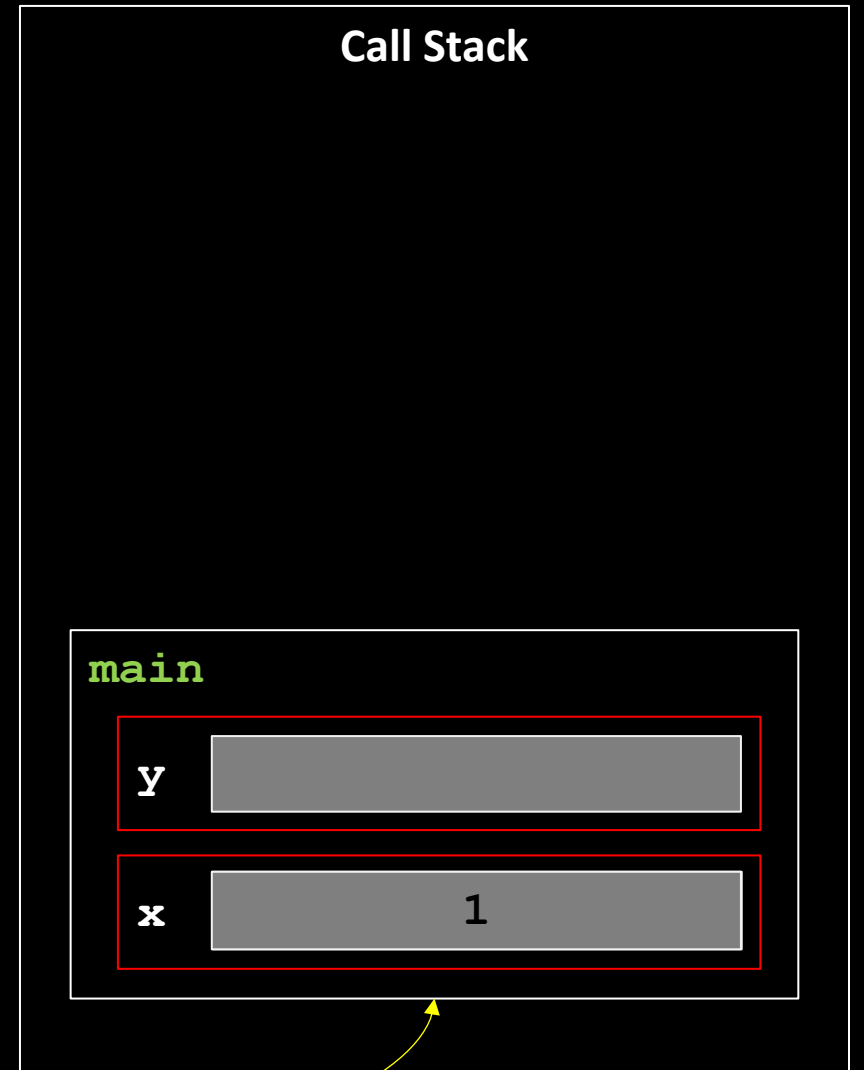
# Homework for Tutorial/Lab 4

- Problem Sets 12 and 13
- Programming Exercises

# Call Stack

```
int main()  
{  
    long x = 1;  
    long y;  
}
```

```
int main()  
{  
    long x = 1;  
    long y;  
}
```



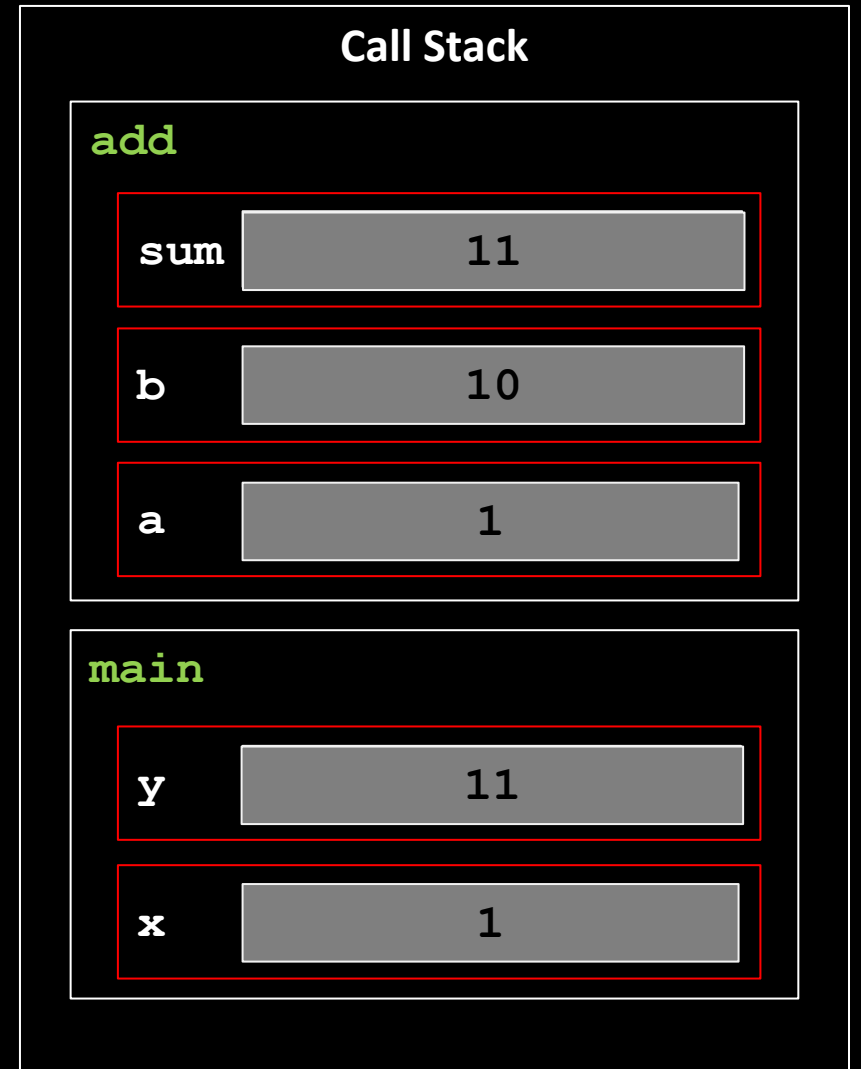
a stack frame

```
long add(long a, long b)
{
    long sum;
    sum = a + b;
    return sum;
}
```

```
int main()
{
    long x = 1;
    long y;
    y = add(x, 10);
}
```

```
long add(long a, long b)
{
    long sum;
    sum = a + b;
    return sum;
}

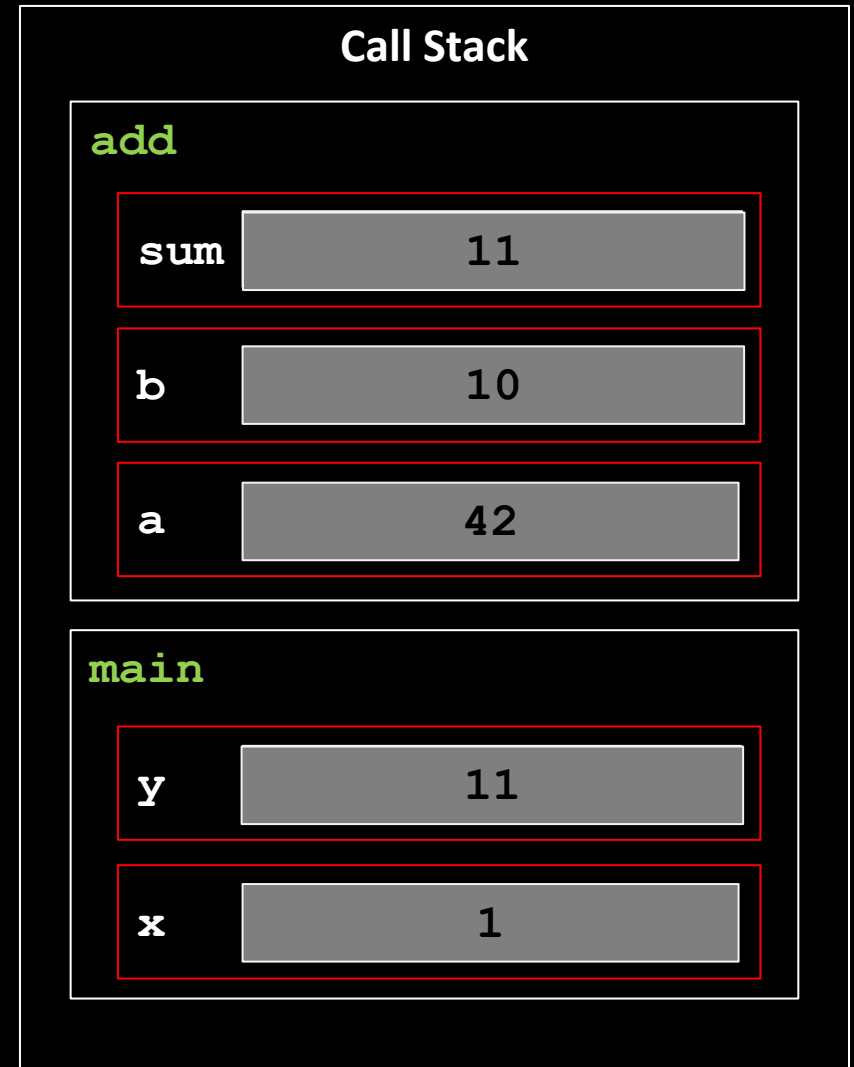
int main()
{
    long x = 1;
    long y;
    y = add(x, 10);
}
```



```
long add(long a, long b)
{
    long sum;
    sum = a + b;
    a = 42;
    return sum;
}
```

Describe the call stack as this instruction is executed

```
int main()
{
    long x = 1;
    long y;
    y = add(x, 10);
}
```

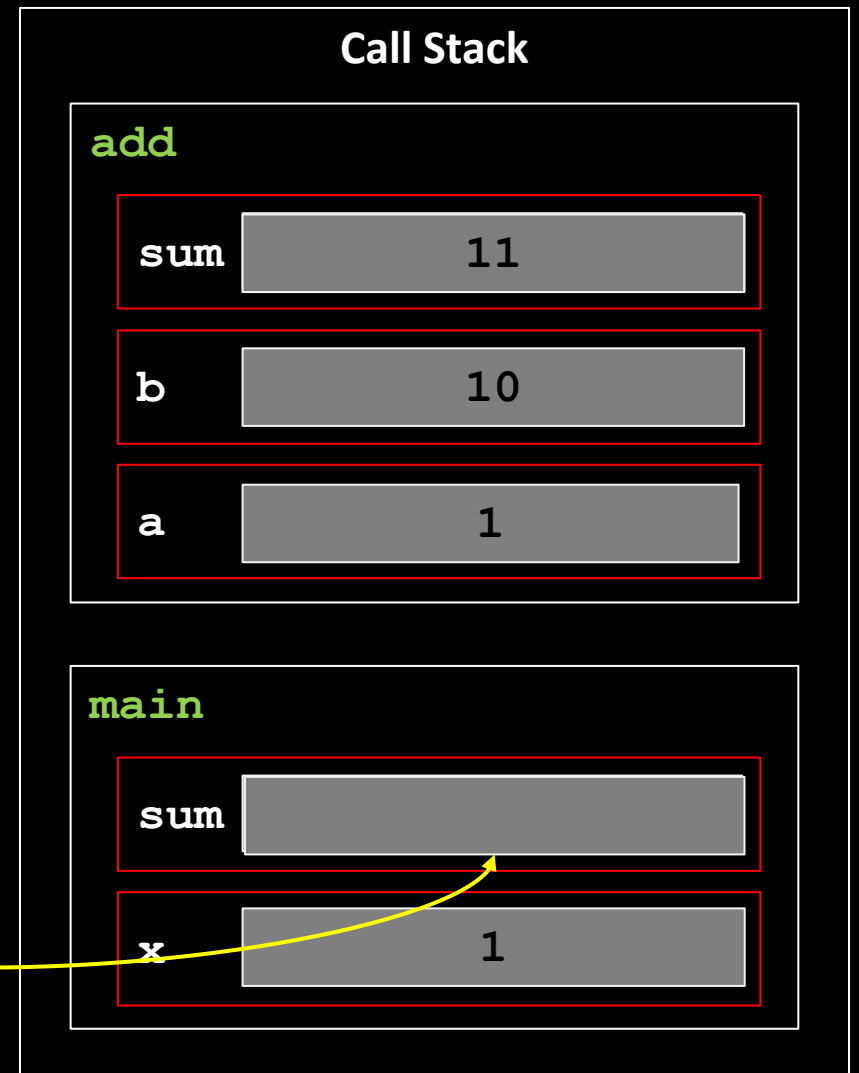


```
void add(long a, long b, long sum)
{
    sum = a + b;
}
```

```
int main()
{
    long x = 1;
    long sum;
    add(x, 10, sum);
}
```

Describe the call stack right after this instruction is executed

The computation to determine the sum was not assigned!

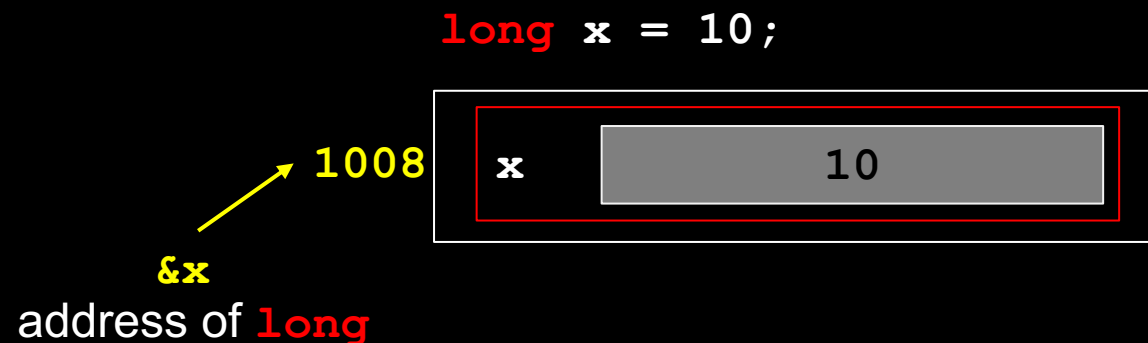




# Pointers

# “Address of” operator: &

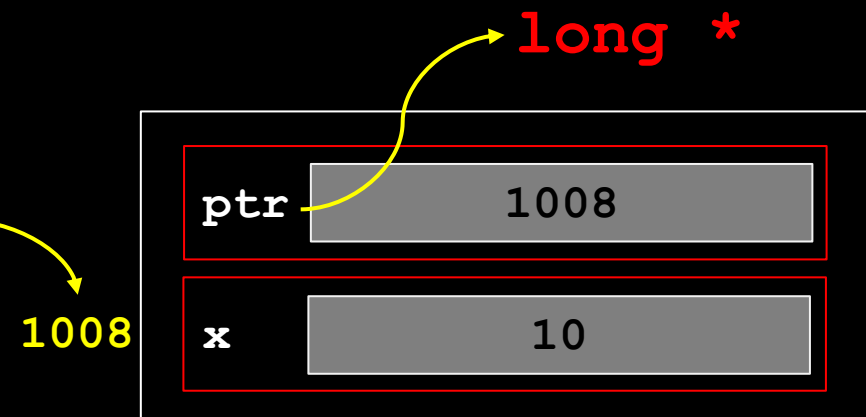
- The value for every **variable** is stored at **some location in memory**
- Given a variable **x**:
  - **x** gives us the value
  - **&x** gives us the **address in memory** where the value for **x** is stored
  - **&x** has type:
    - “**address of T**”
    - where **T** is the **type of x**



# “Pointer” type: `<type> *`

- We can define a **variable** that **stores** the **memory address** of a specific **C type value**
  - For example:

```
long x = 10;  
long * ptr = &x;
```



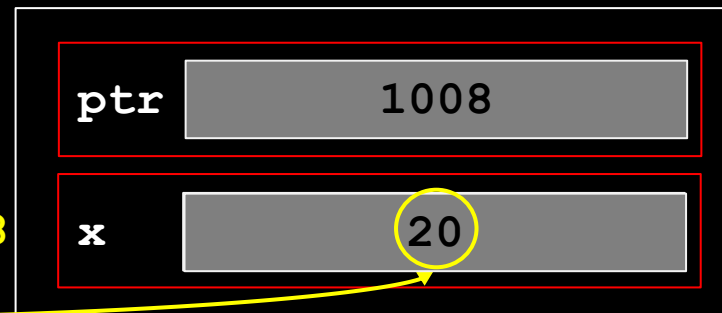
# “Pointer” operator: \*

- The **value** stored **at the memory address of a pointer**, may be **accessed** by using the **\*** operator
- Given a pointer **ptr**:
  - **ptr** gives us a memory address
  - **\*ptr** allows us access the value stored at that **memory address**

- For example:

- `long x = 10;`
- `long * ptr = &x;`
- `*ptr = 20;`

1008

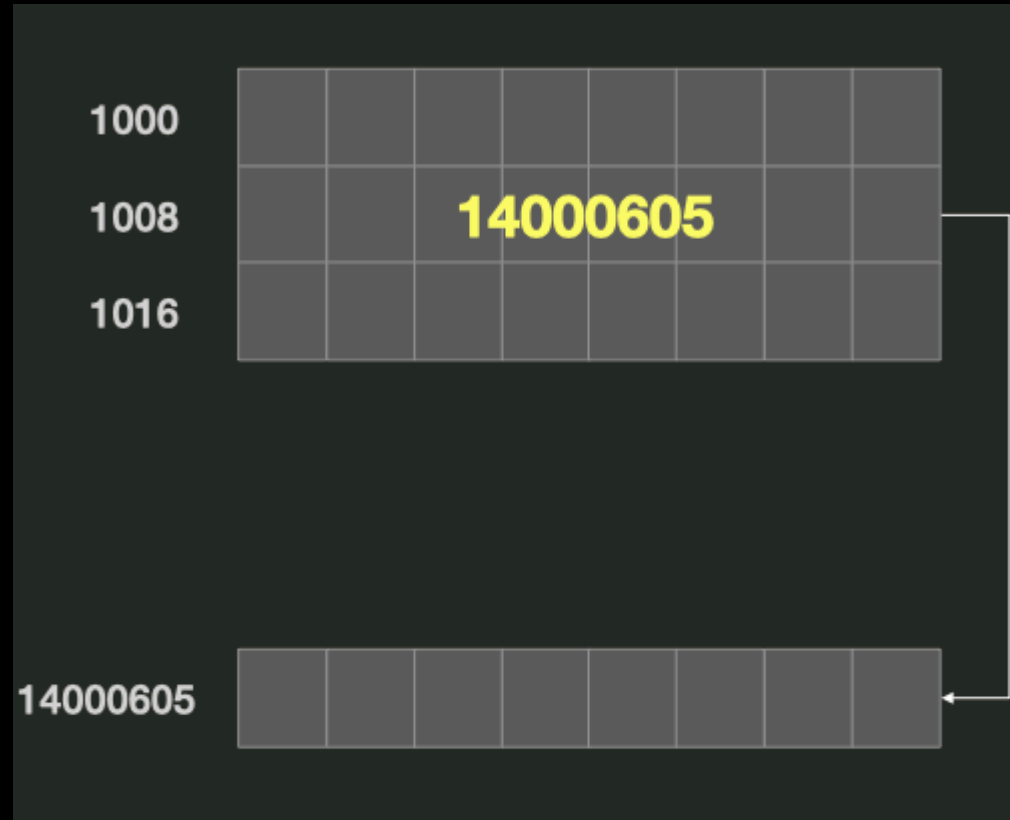


```
#include "cs1010.h"

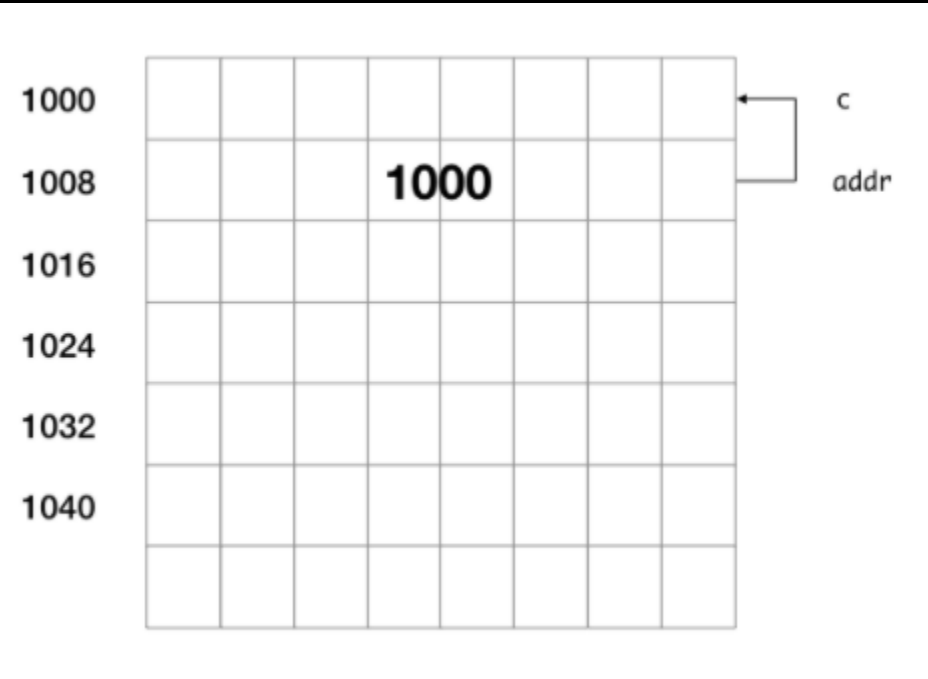
void add(long a, long b, long sum)
{
    sum = a + b;
    cs1010_println_long((long) &sum);
}

int main()
{
    long x = 1;
    long sum;
    add(x, 10, sum);
    cs1010_println_long((long) &sum);
}
```

```
int main()
{
    double * ptr;
    *ptr = 1.0;
}
```



```
int main()
{
    double c;
    double * ptr;
    ptr = &c;
    *ptr = 1.0;
}
```



# Rules on pointers

- A *T* pointer, where *T* is some C type, can only point to a variable of type *T*

- Example:

```
double pi = 3.1415926;
```

```
long radius = 5;
```

```
double *addr;
```

```
addr = &pi; // ok
```

```
addr = &radius; // not ok (radius is a long)
```



# Rules on pointers

- We cannot change the address of a variable, but we can change what a pointer is pointing at
- Example:

```
long x = 1;
```

```
long y = 2;
```

```
&x = &y;    // invalid
```

# Rules on pointers

- We can perform **add** and **subtract on pointers**
  - This changes the address by 1 unit, where this unit corresponds to the length (i.e., number of bytes) of the type

- Example:

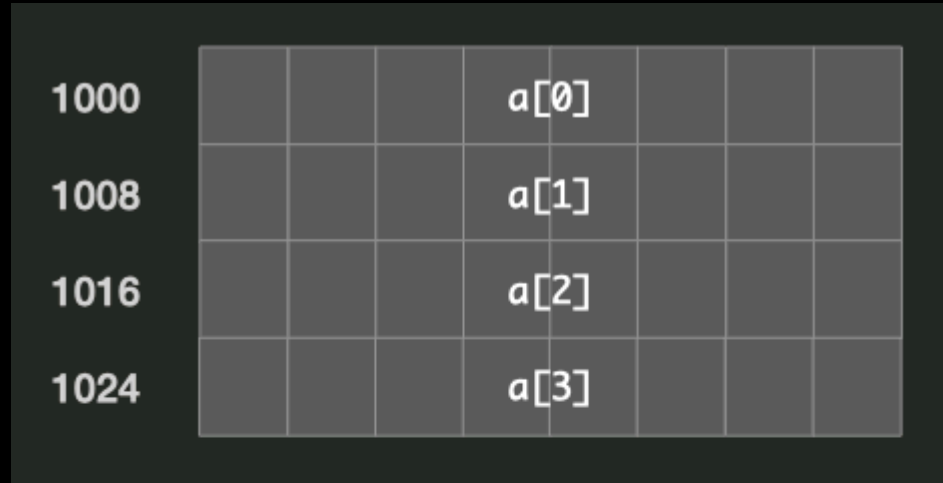
```
long x = 1;  
long * ptr = &x; // assume &x is 2404  
ptr += 1;        // ptr now points at? 2412
```

# Arrays

Array is a *compound data type* that **stores multiple values of the same type**

Example:

```
long a[4];
```



This statement declares an array of 4 long values

Arrays **contiguous** sequence of memory

Example:

```
long a[10]; // declare  
a[0] = 8; // write 1st element  
a[4] = 100; // write 5th element
```

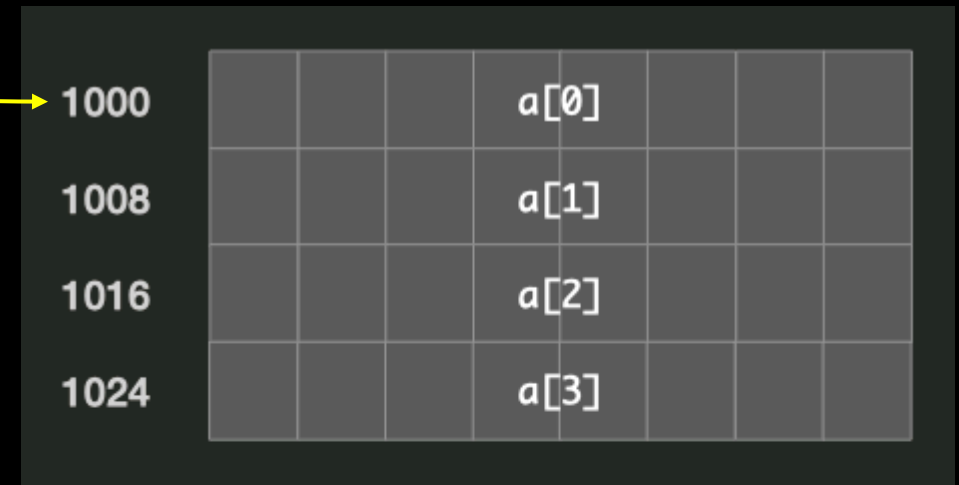
**a[i]** accesses the **i+1**th element of the array

Recall that **i** starts at 0

# Array decay


- Consider `long a[4];`
- `a` decays to `&a[0]`

notice that the address of any element is simply `&a[i]`, or simply the **starting address of the array** +  $(i * unit)$



# Arrays as a parameters

```
// function declaration
long max(long list[], long len) {
    :
}
:
int main() {
    long marks[10]; // declaration - empty []
    max(marks, 10); // invoking max; marks as parameter
}
&marks[0]
```





# Using an array as a lookup table

```
// Exercise 1: Days
long days(long month)
{
    long days_in_month[12] = {31, 28, 31, 30, 31, 30,
                              31, 31, 30, 31, 30, 31};

    long days_since = 0;
    for (long i = 0; i < month - 1; i += 1) {
        days_since += days_in_month[i];
    }
    return days_since;
}
```

## Using an array as a list

```
long max(long list[], long length)
{
    long max_so_far = list[0];
    for (long i = 1; i != length; i += 1) {
        if (list[i] > max_so_far) {
            max_so_far = list[i];
        }
    }
    return max_so_far;
}
```

# Array issues: comparison & assignment

```
long a[2] = {0, 1};  
long b[2] = {0, 1};  
if (a == b) { // always false  
    :  
}  
b = a; // not possible
```

# Array issues: index out of bounds

```
int main()
{
    long a[10];
    for (long i = 0; i <= 10; i += 1) {
        a[i] = 1;
    }
}
```

# Other details about arrays in C

- Including:
  - array initialisation
  - avoiding the use of variable-length arrays
  - determining the size of the array
  - how to read arrays with CS1010 I/O library

# Strings

# Strings in C

- Just an array of char values
- Always **terminated by** a value **0** (aka **null** or **\0**)

# Special Characters

- `\0`: null character
- `\n`: new line
- `\t`: tab
- e.g.,  
`cs1010_println_string("group\tname\nC03\tJohn\n");`



# String Literals

- **Unmodifiable** string between two double-quotes.
- Stored not on the stack but in a **read-only region of memory**.

```
char *str1 = "hello!"  
str1[0] = 'j'; // error
```

**str1** is a pointer on the stack, pointing to a **read-only memory**.

# Common String Bugs I

```
char str2[7] = "hello!" // char array; not pointer  
str2[0] = 'j'; // ok
```

**str2[7]** is an array and contains a copy of the string "hello" on the stack.

# Common String Bugs II

```
char src[6] = "hello!"  
char dst[6];  
for (long i = 0; i < 6; i += 1) {  
    dst[i] = src[i];  
}
```

## Common String Bugs II

```
char src[6] = "hello!"  
char dst[6];  
strcpy(dst, src);
```