CS2100 - L04 - Functions and Pointers Week 2+3

4.1 - Functions

- Syntax and Semantics

4.2 - Pointers

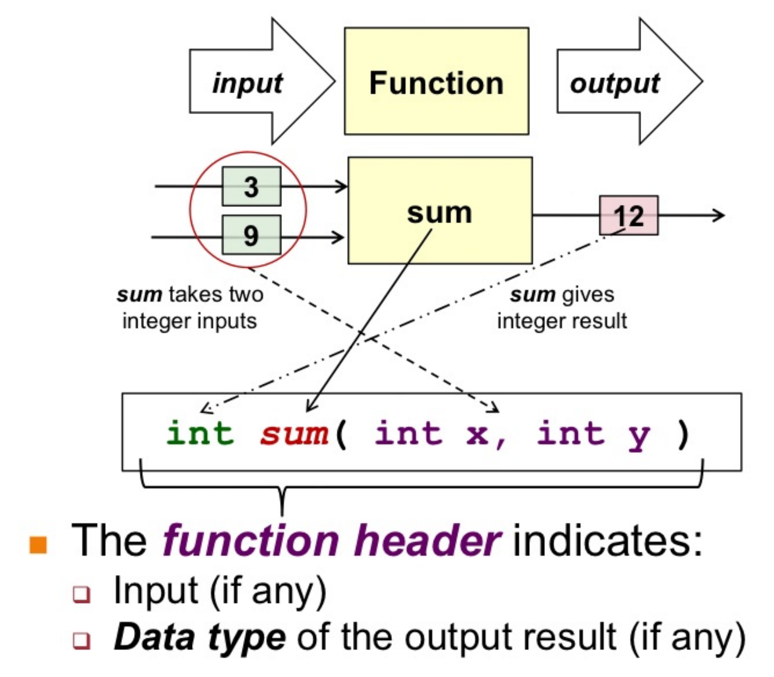
- Syntax and Semantics

4.3 - Function Parameter Passing

- By Value
- By Address / Pointer

4.1 – Functions

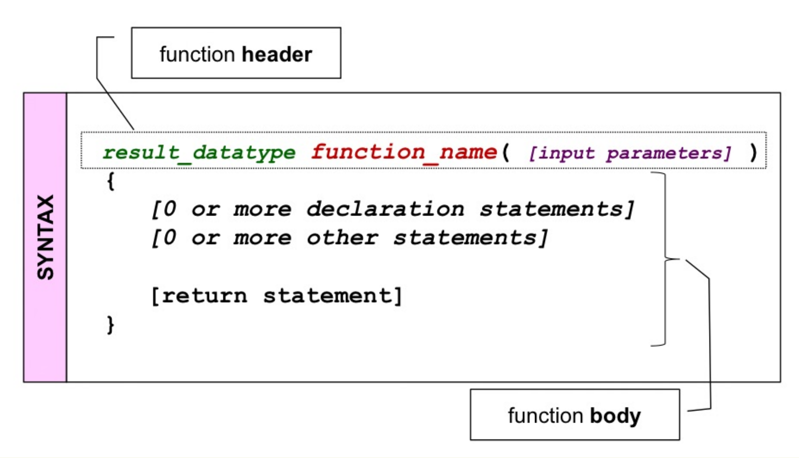
- Large programs should be modularized
  - Python, JavaScript, C, etc. : function
  - Java, C++ : method
- Functions allow :
  - better maintenance
  - reusability



```
int sum( int x, int y )
```

- The *function header* indicates:
  - Input (if any)
  - ***Data type*** of the output result (if any)

Syntax



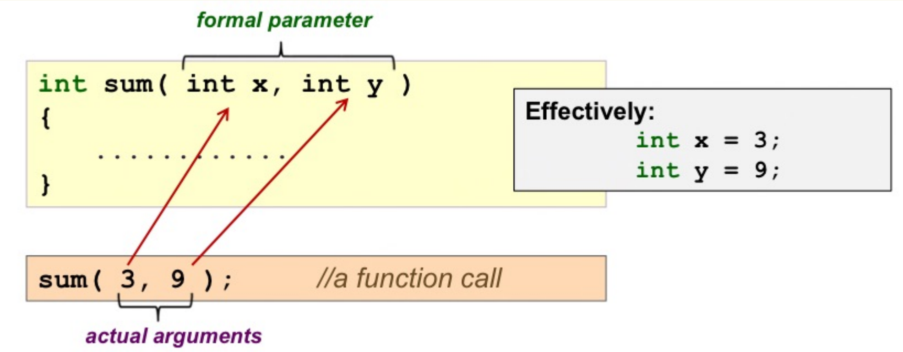
function **header**

SYNTAX

```
result_datatype function_name( [input parameters] )
{
    [0 or more declaration statements]
    [0 or more other statements]

    [return statement]
}
```

function **body**

Function calls — parameters and arguments



formal parameter

```
int sum( int x, int y )
{
    ............
}
```

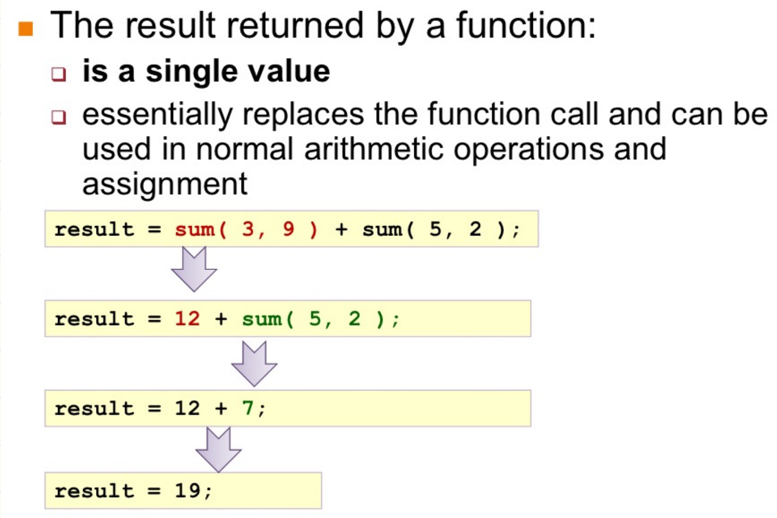
```
sum( 3, 9 );        //a function call
```

actual arguments

Effectively:

```
int x = 3;
int y = 9;
```

Function calls — returned result



- The result returned by a function:
  - **is a single value**
  - essentially replaces the function call and can be used in normal arithmetic operations and assignment

```
result = sum( 3, 9 ) + sum( 5, 2 );
```

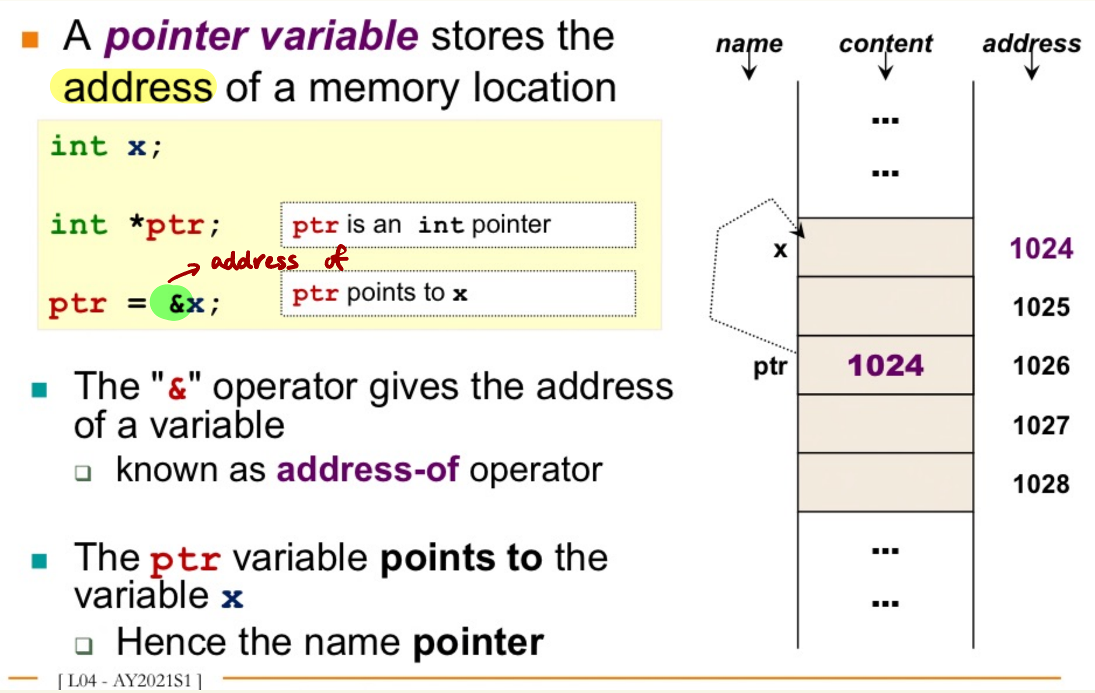
```
result = 12 + sum( 5, 2 );
```

```
result = 12 + 7;
```

```
result = 19;
```

4.2 – Pointers

Pointers – declaration



- A *pointer variable* stores the address of a memory location

```
int x;

int *ptr;

ptr = &x;
```

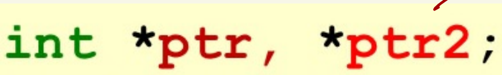
ptr is an int pointer

address of

ptr points to x

- The "&" operator gives the address of a variable
  - known as **address-of** operator
- The ptr variable **points to** the variable x
  - Hence the name **pointer**

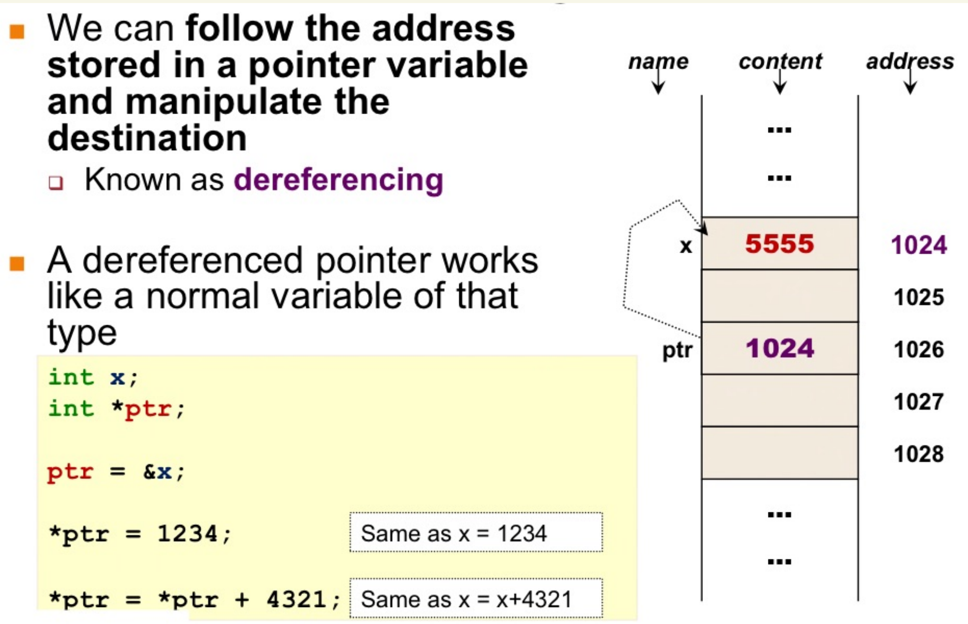
name	*content*	*address*
	...	
	...	
x		1024
		1025
ptr	1024	1026
		1027
		1028
	...	
	...	



```
int *ptr, *ptr2;
```

≠ int* ptr1, ptr 2;

Pointers – dereferencing



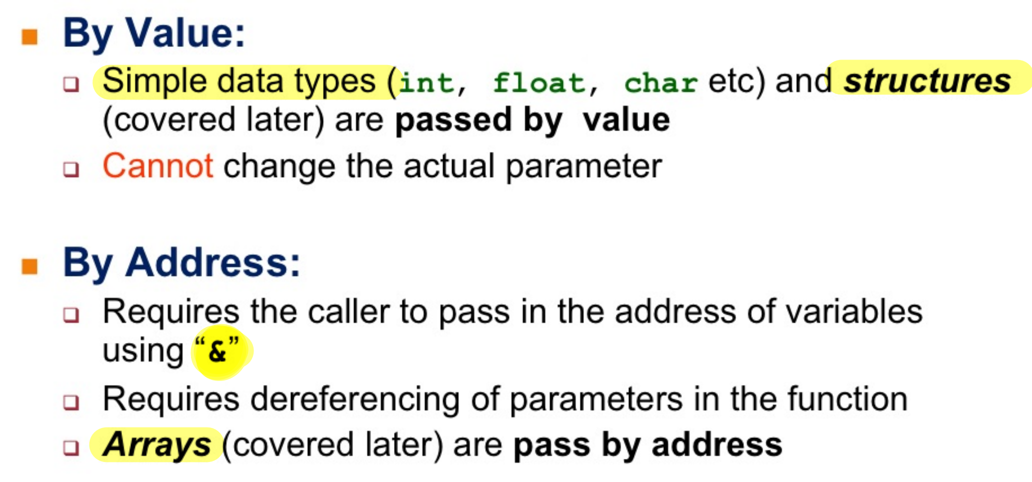
- We can **follow the address stored in a pointer variable and manipulate the destination**
  - Known as **dereferencing**
- A dereferenced pointer works like a normal variable of that type

```
int x;
int *ptr;

ptr = &x;

*ptr = 1234;              Same as x = 1234

*ptr = *ptr + 4321;  Same as x = x+4321
```

* notation :

- Declaration : declare pointer variable
- Usage : dereference pointer variable

4.3 - Function Parameter Passing



- **By Value:**
  - Simple data types (`int`, `float`, `char` etc) and ***structures*** (covered later) are **passed by value**
  - Cannot change the actual parameter

- **By Address:**
  - Requires the caller to pass in the address of variables using **“&”**
  - Requires dereferencing of parameters in the function
  - ***Arrays*** (covered later) are **pass by address**