UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

POINTERS AND DYNAMIC MEMORY MANAGEMENT I

BEESHANGA ABEWARDANA JAYAWICKRAMA

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

UTS CRICOS PROVIDER CODE: 00099F

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

SCOPE OF VARIABLES

UTS:
ENGINEERING AND INFORMATION TECHNOLOGY

SCOPE OF VARIABLES

Scope of a variable: the part of the program where a variable name is known

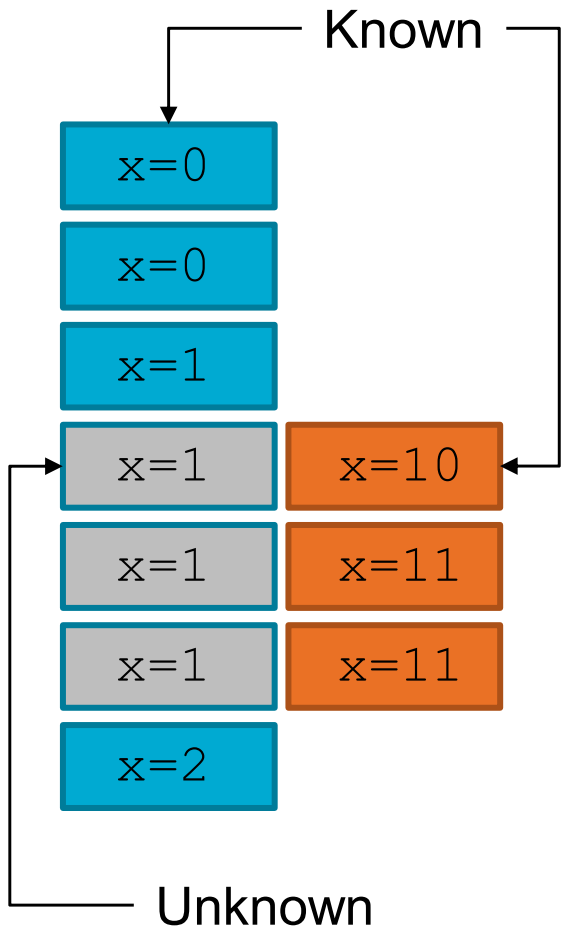
C has block scope

```
01   if(/*condition*/)
02   {
03     int u = 4;
04     int v = 8;
05     ...
06     ...
07   }
08   else
09   {
10     ...
11     ...
12   }
```

Line	U	V
01	Unknown	Unknown
02	Unknown	Unknown
03	Known	Unknown
04	Known	Known
05	Known	Known
06	Known	Known
07	Known	Known
08	Unknown	Unknown
09	Unknown	Unknown
10	Unknown	Unknown
11	Unknown	Unknown
12	Unknown	Unknown

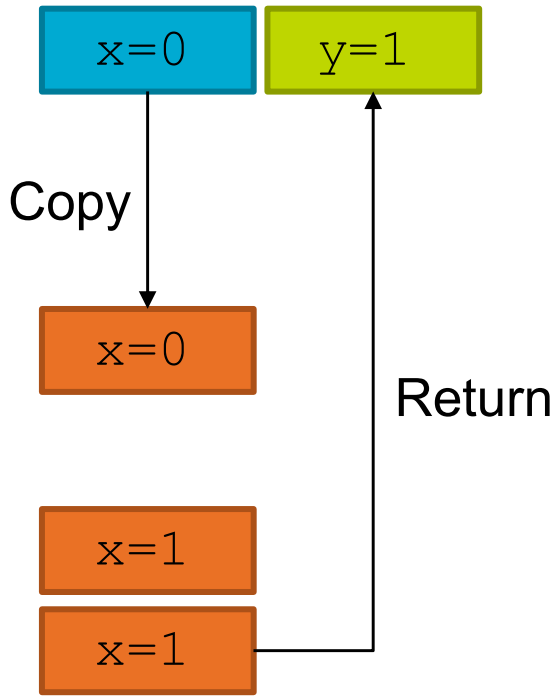
SCOPE OF VARIABLES WITH THE SAME NAME

```
int main(void)
{
  int x = 0;
  {
    x++;
    int x = 10;
    x++;
  }
  x++;
  return 0;
}
```



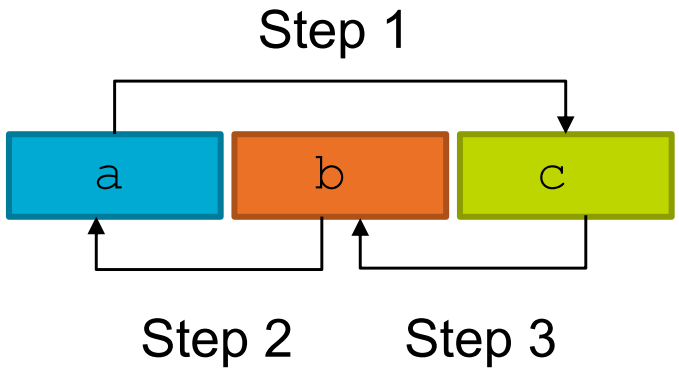
SCOPE OF VARIABLES PASSED TO FUNCTIONS

```
int main(void)
{
   int x = 0;
   int y = fun(x);
   return 0;
}
int fun(int x)
{
   x++;
   return x;
}
```



SWAP TWO INTEGERS

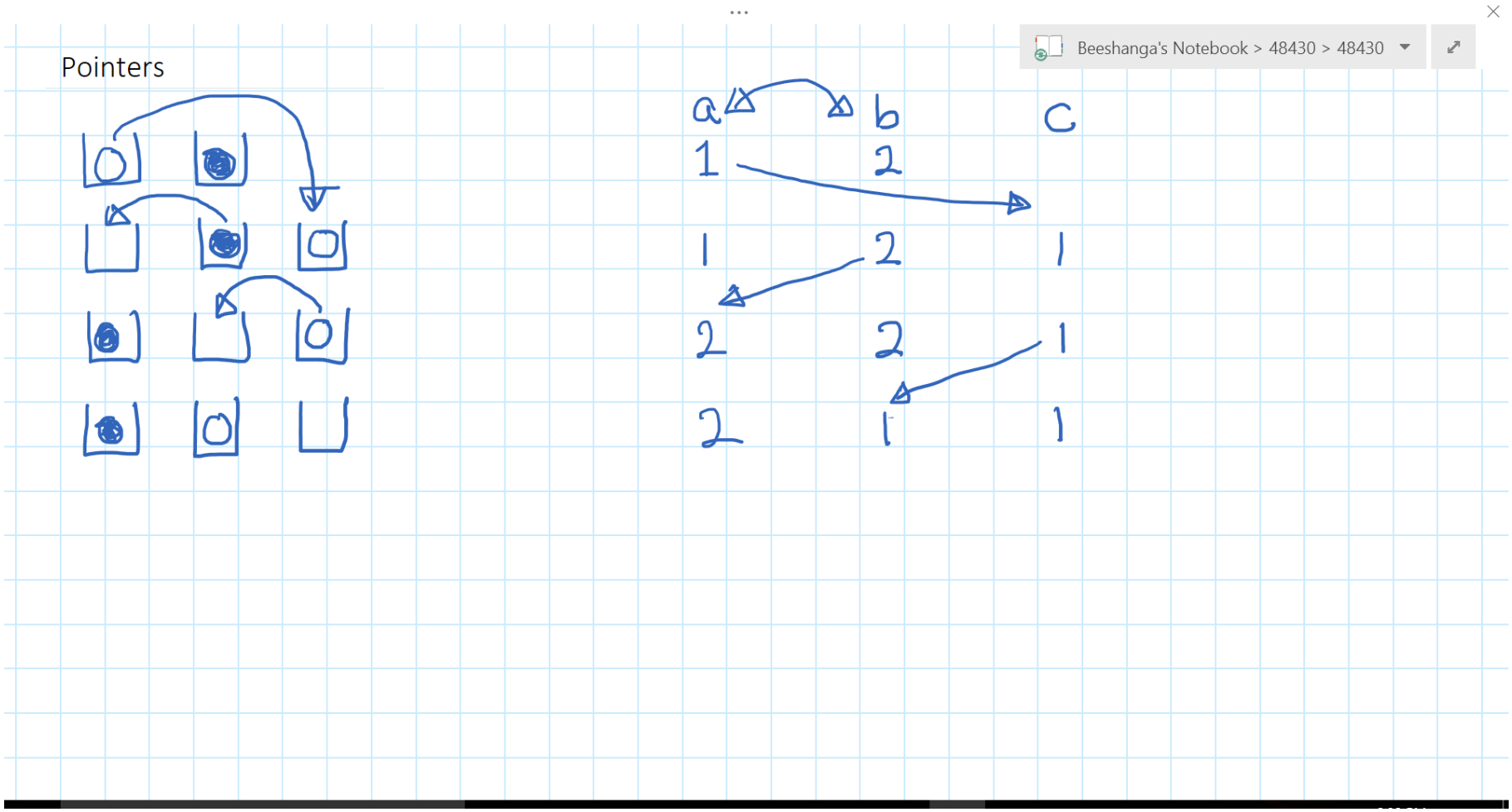
```
int a = 1, b =2;

int c = a;

a = b;

b = c;
```



Try the above code inside the main function.

Write a function to swap two integers. Is that as simple as copy paste the above code to a function?

SWAP TWO INTEGERS



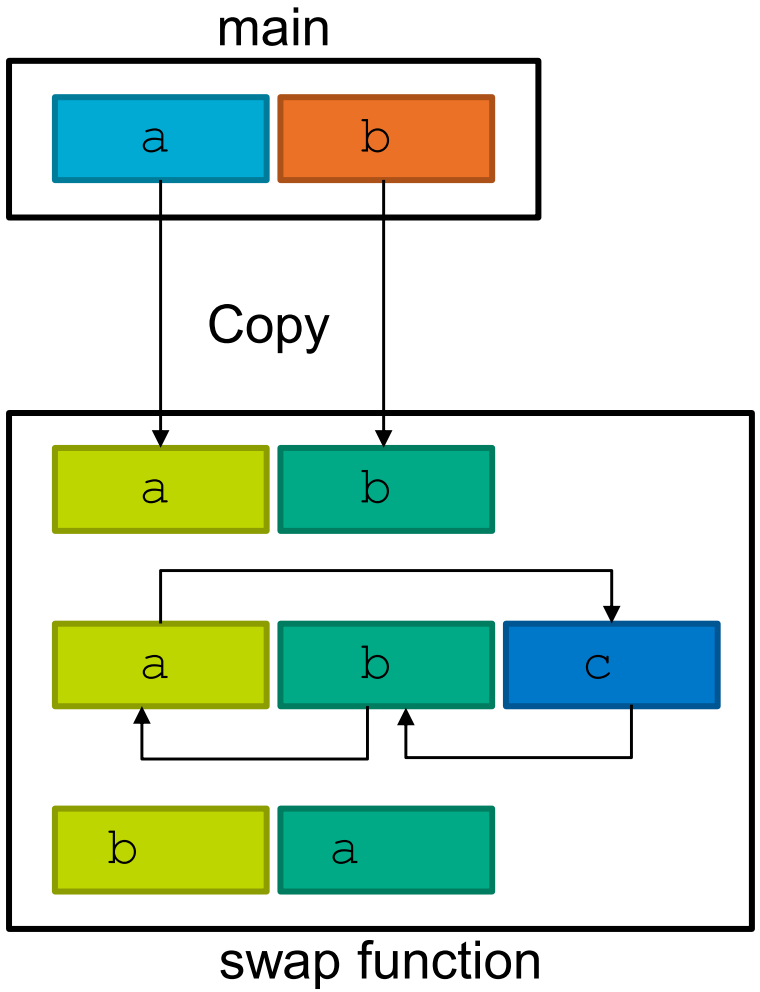
SWAP FUNCTION – DOESN'T WORK

```
int a = 1, b =2; /*main*/

swap(a,b); /*main*/


void swap(int a, int b);
```

Function copies the variable values from main.

Function swaps only the two local copies of `a` and `b`. Variables in the main are not changed.

Function does NOT swap `a` and `b` values in main.



UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

POINTERS AND DYNAMIC MEMORY MANAGEMENT I

BEESHANGA ABEWARDANA JAYAWICKRAMA

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

UTS CRICOS PROVIDER CODE: 00099F

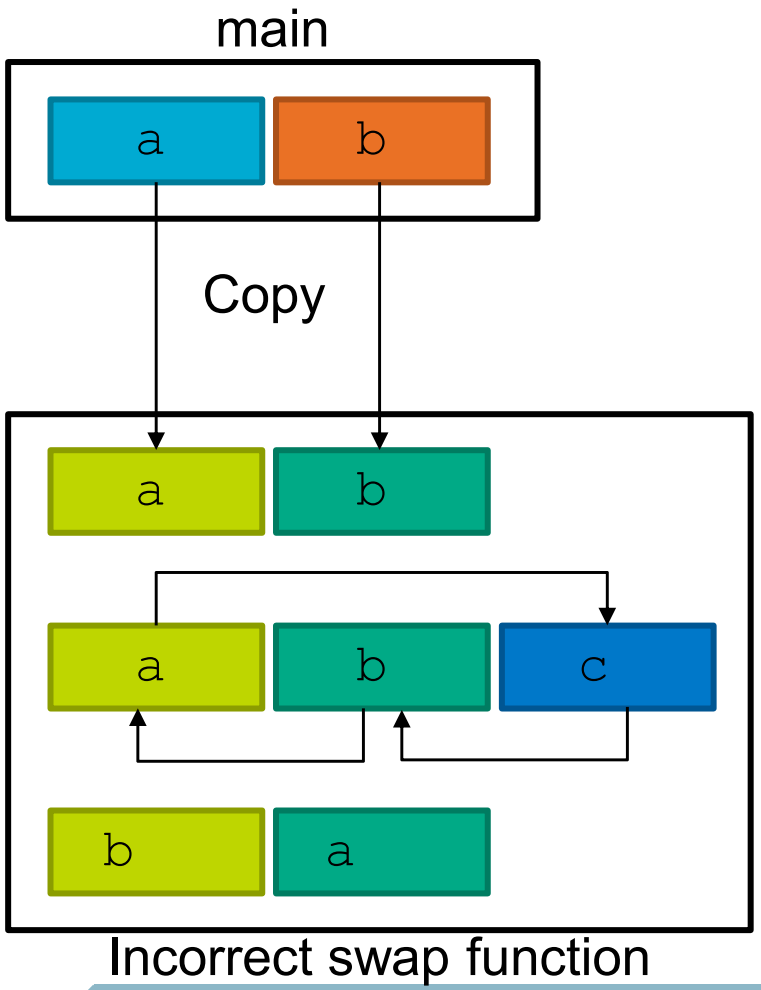
UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

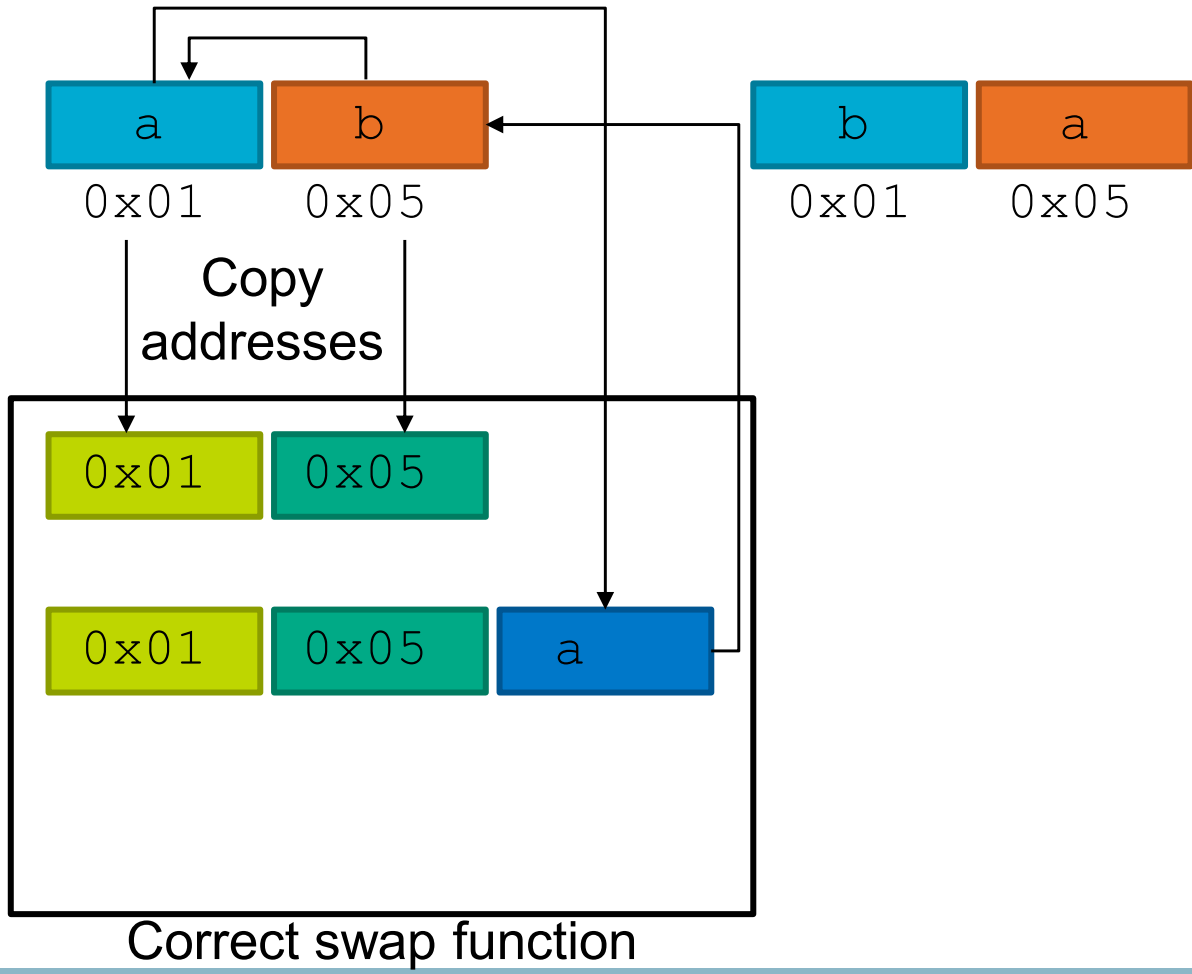
POINTERS

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

FIXING THE SWAP FUNCTION – LOGIC

Sometimes knowing the value of a variable is not sufficient. Sometimes we need to know the memory location where a variable is stored.





POINTERS

A pointer is a variable which contains a **memory location** (of another variable). Some notations to be familiar with as follows.

`int x` – define a new variable that contains an integer

> `x` – a value

> `&x` – memory location of the variable `x`

`int* xp` – define a new variable that contains the address of an integer

> `xp` – a memory location (address)

> `*xp` – get the value stored at the memory location pointed by `xp`

`void fun(double* yp)` – function expects memory location of a double

> `double d; fun(&d)` – pass the memory location of `d` to `fun`

FIXING THE SWAP FUNCTION – IMPLEMENTATION USING POINTERS

```
void swap(int* ap, int* bp)
{
  int c = *ap;
  *ap = *bp;
  *bp = c;
}
```

Main can call the swap function as

```
int a = 1, b = 2;
swap(&a,&b);
```

Good coding habit: Pointer variable names conventionally have `p` in the end. E.g `int* ap`, not `int* a`

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

POINTERS AND DYNAMIC MEMORY MANAGEMENT I

BEESHANGA ABEWARDANA JAYAWICKRAMA

UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

UTS CRICOS PROVIDER CODE: 00099F

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

POINTERS IN FUNCTIONS AND ARRAYS

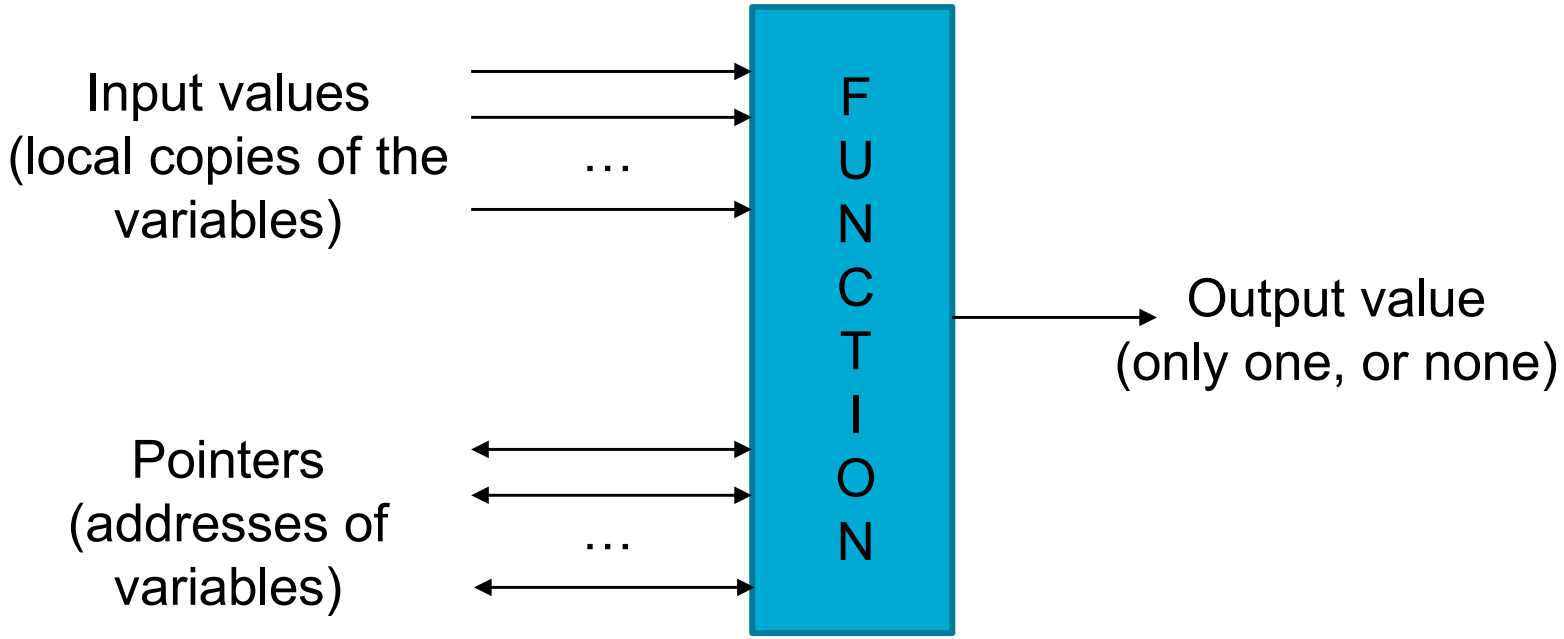
UTS:
ENGINEERING AND
INFORMATION
TECHNOLOGY

UTS CRICOS PROVIDER CODE: 00099F

RETURNING FROM FUNCTIONS

A function can explicitly return only ONE value.

However through pointers, a function can indirectly input/output any number of values.



POINTERS AND ARRAYS

Arrays and pointers are similar.

Consider the following array definition.

```
int a[] = {0, 1, 2};
```

`a` is a pointer to the 0th element in the array (`a` is an address)

Note the following:

`&a[0] = a`

`a[1] = *(a+1)` – provides an alternative way to travel through an array

UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

DYNAMIC MEMORY MANAGEMENT

UTS:
ENGINEERING AND INFORMATION TECHNOLOGY

POINTERS

A pointer is nothing but a memory address

Pointers enable dynamic memory management, i.e.

> Dynamic memory allocation

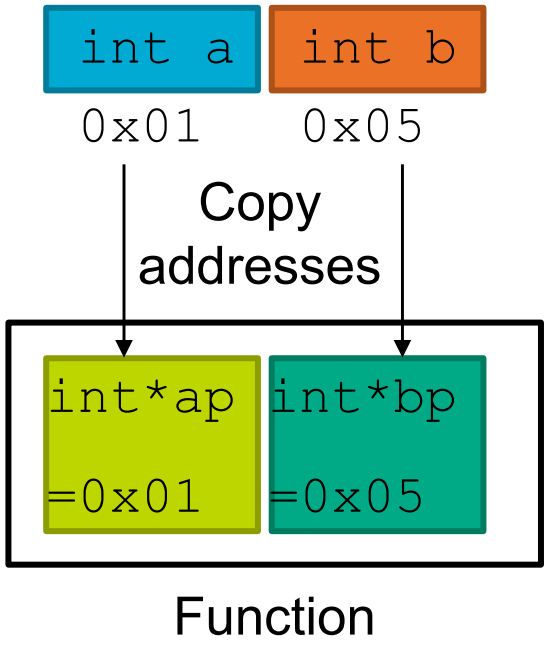
> Dynamic memory deallocation

Why do we need dynamic memory management?

Sometimes we don't know the required size of memory. Enables efficient usage of memory resources in runtime.

> Arrays allow static memory usage (fixed size). What if the size of the array needs to be determined in runtime?

POINTERS IN FUNCTIONS VS DYNAMIC MEMORY ALLOCATION




```
#include <stdlib.h>


int* ap;

ap =/* allocate 4n bytes */
```

Pointers in the function contain the addresses of integers that already exist (point to pre-allocated memory)

In dynamic memory allocation, we want to allocate new memory to a pointer.

DYNAMIC MEMORY ALLOCATION

```
void *malloc(size_t size);
```

- Allocate a block of *size* bytes (not initialised), return a pointer to the beginning of the block.
- Void pointers are generic pointers that can point to anything. They must be type casted.

- Returns a NULL pointer if the dynamic memory request cannot be granted.

- Other options are: `calloc` and `realloc` – for selfstudy

DYNAMIC MEMORY ALLOCATION - EXAMPLE

```
int* xp;

xp = (int*) malloc(1*sizeof(int));

if (xp==NULL){

    printf("Allocation failed.\n");

}
```

Note:

- Type casting.

- How the size of the allocation was calculated.

- NULL test.

DYNAMIC MEMORY DEALLOCATION

```
void free(void *ptr);
```

- A block of memory previously allocated using malloc is made available for other allocations.

- It does not change ptr variable.

- It is good coding practice to change ptr to NULL after calling free().

DYNAMIC MEMORY DEALLOCATION - EXAMPLE

```
/* After the code shown in previous slide, add the
following lines */

free (xp);

xp = NULL;
```

- Only previously allocated memory can be deallocated
- Pointer made NULL after free