DASF004
Basic and Practice in Programming

Lecture 7

Pass by Reference

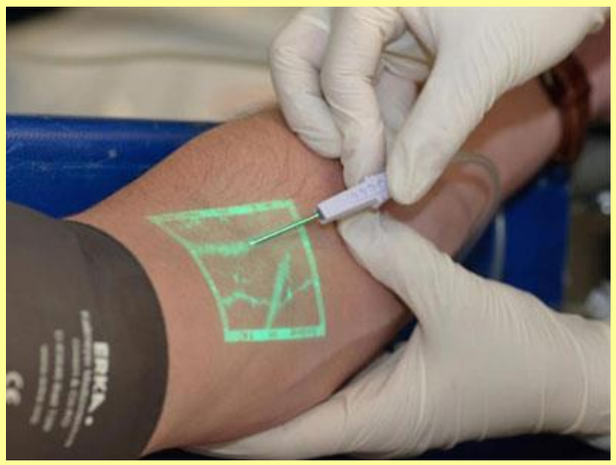
Food for your MIND

- Augmented Reality in Medical Applications
- X-ray vision ……

- Medical Imaging Device
  - X-Ray, MRI, CT Scan, Ultra-sonic, etc
- Overlay medical images on patient in real time

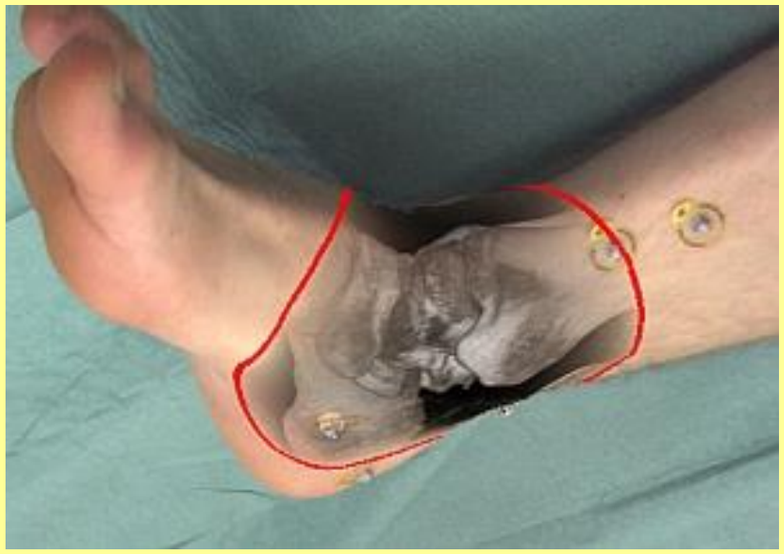
Finding Veins

- Finding veins
  - For blood draw, Injection, Hydodermic
- Sometimes, it is difficult to find the vein of some patients

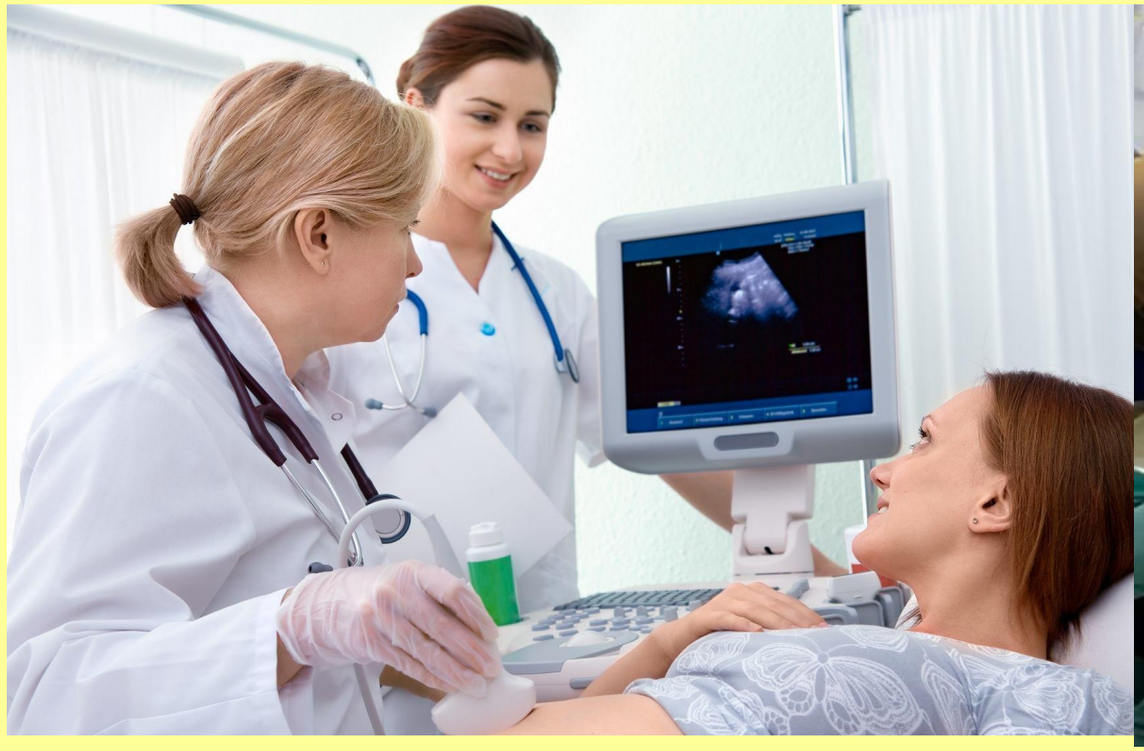


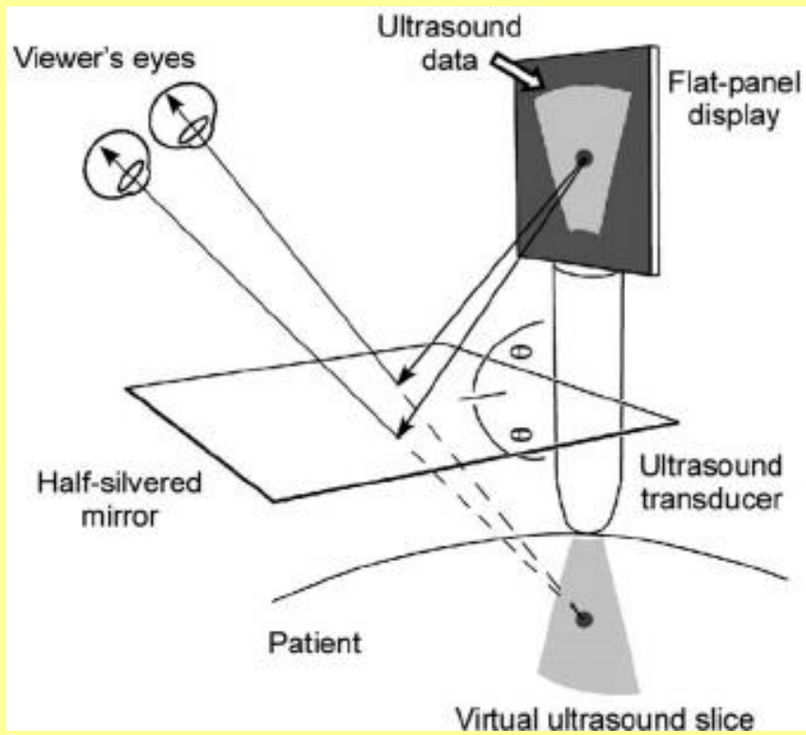
Head-mounted Display

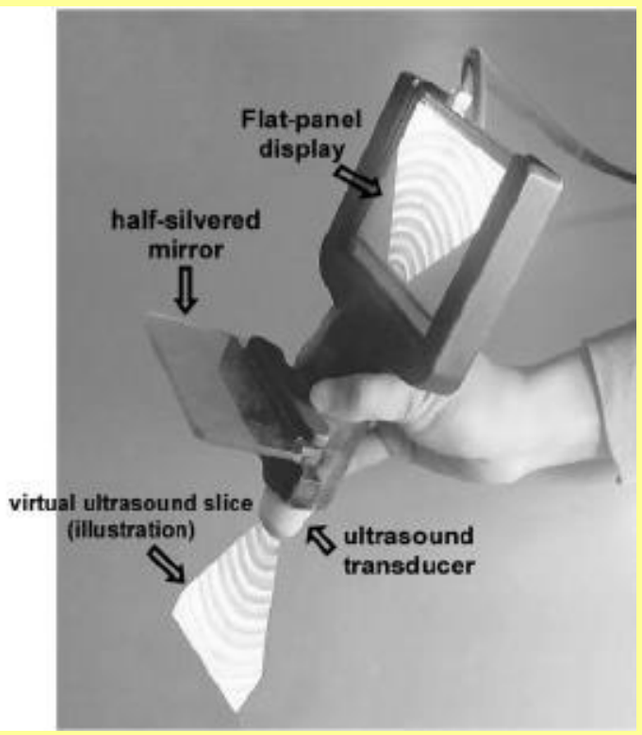
- For medical diagnoses



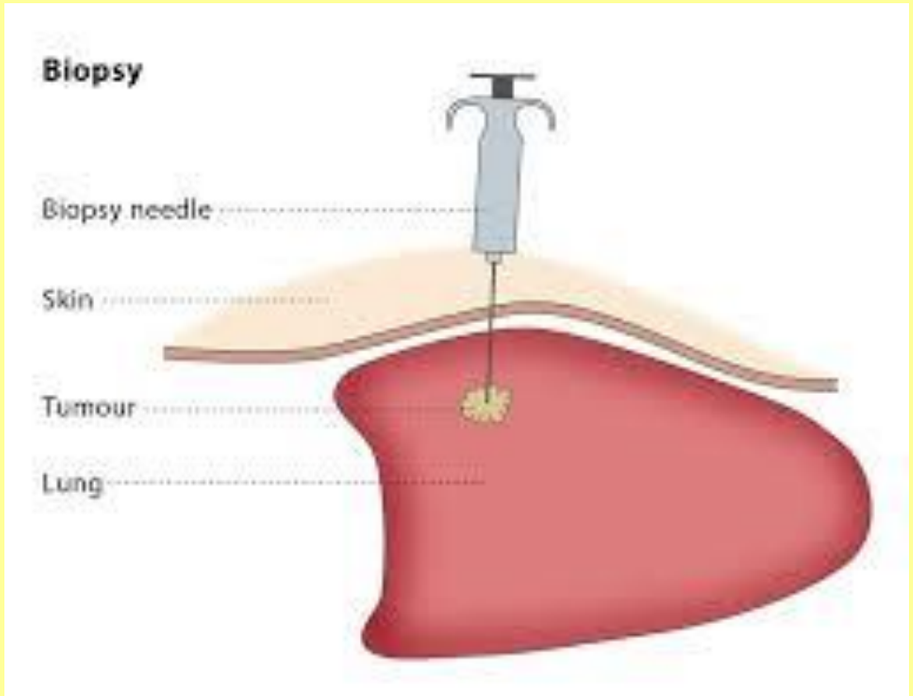
Hand-held Display

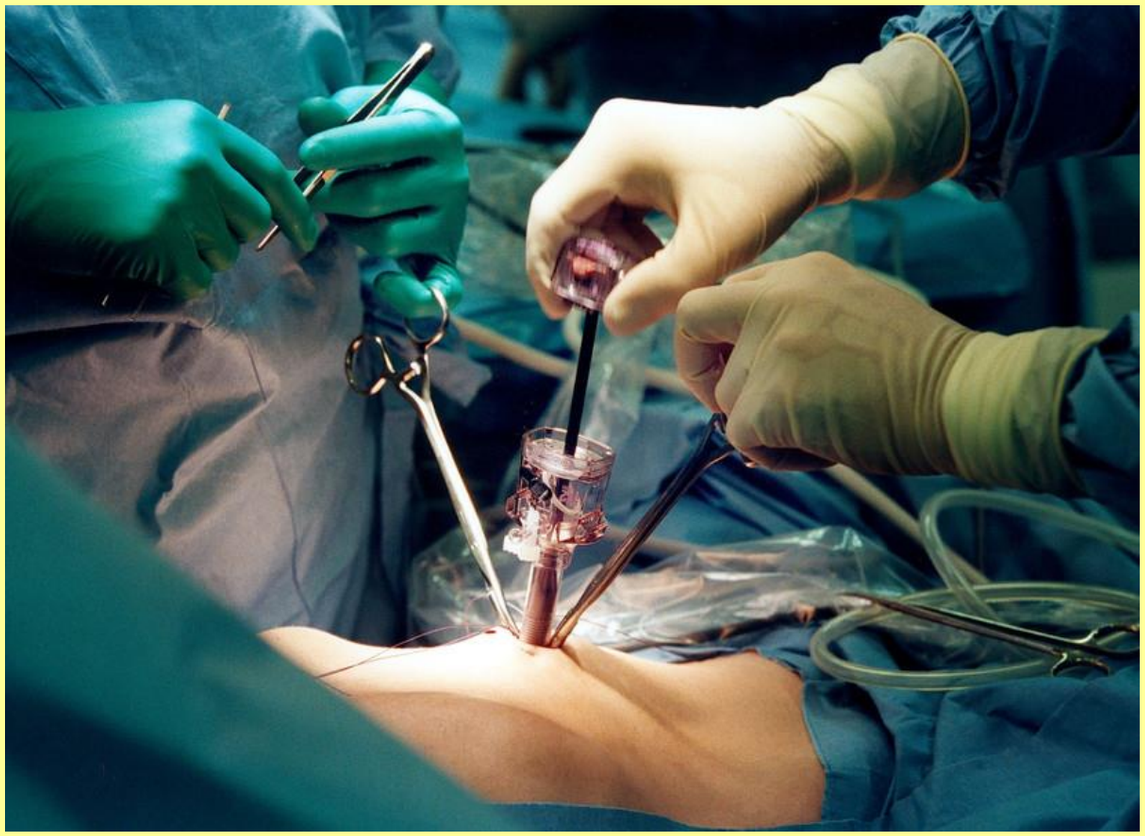


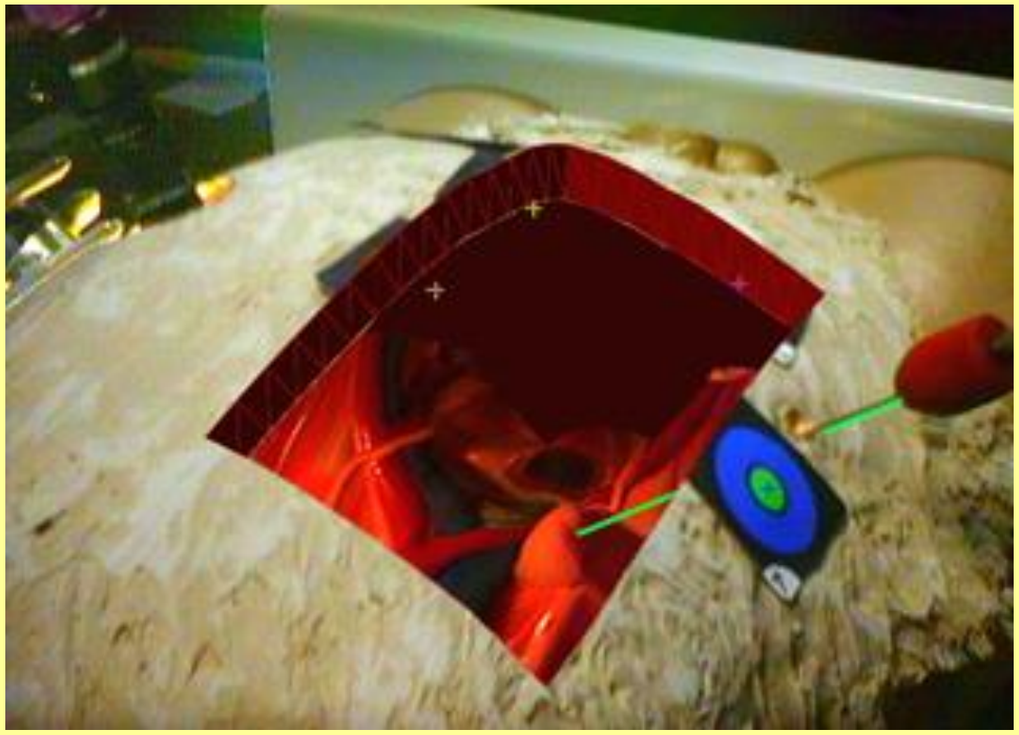




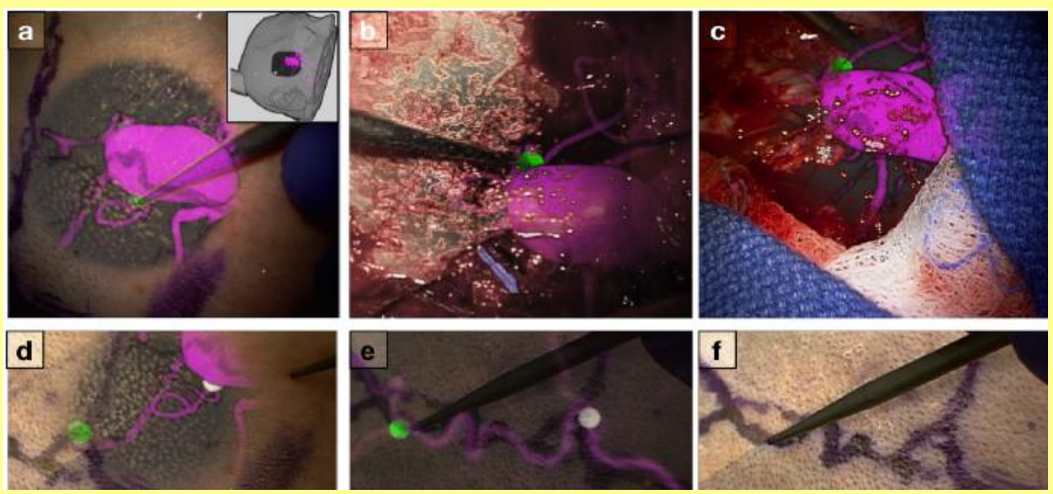
Biopsy and Microscopic Surgery

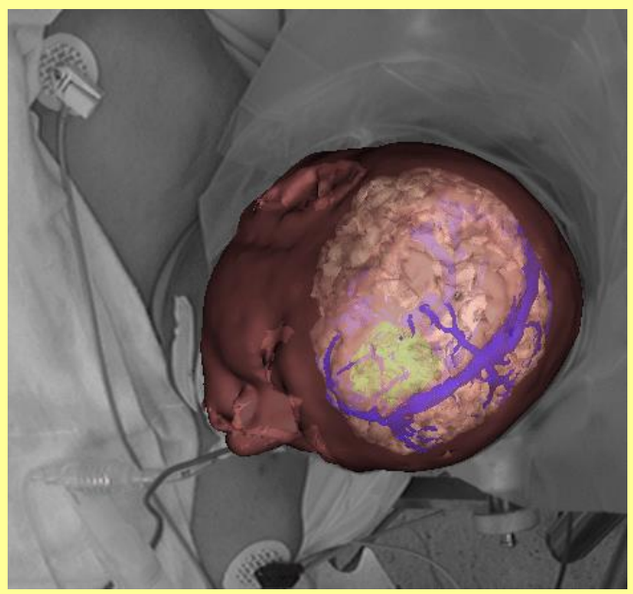






Operating Microscope





Agenda

Pass by reference

Issues in Arrays and Functions:

- **Issue #1:**
- You can only pass individual array items as argument to functions, for example:

```
int A[2] = {0};
int result = MyFunction(A[0],A[1]);
```

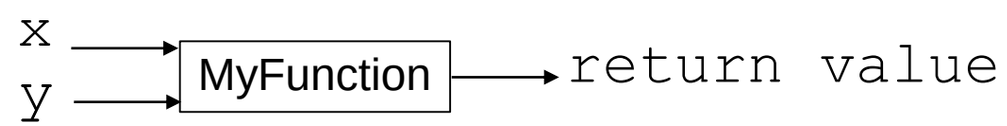
You cannot pass the entire array as argument:

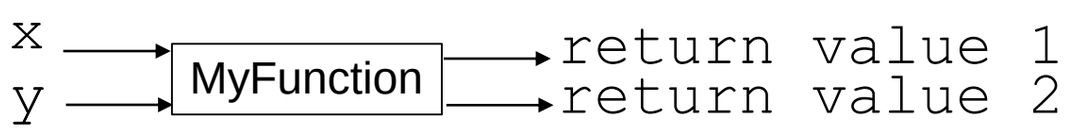
```
int A[2] = {0};
int result = MyFunction2(A);     // WRONG!!!
```

- **Issue #2:**
- You can only return ONE return value in functions, for example:

```
int MyFunction(int x, int y)
{ return x+y;
}
```

What if you want to create a function that returns more than one return value?





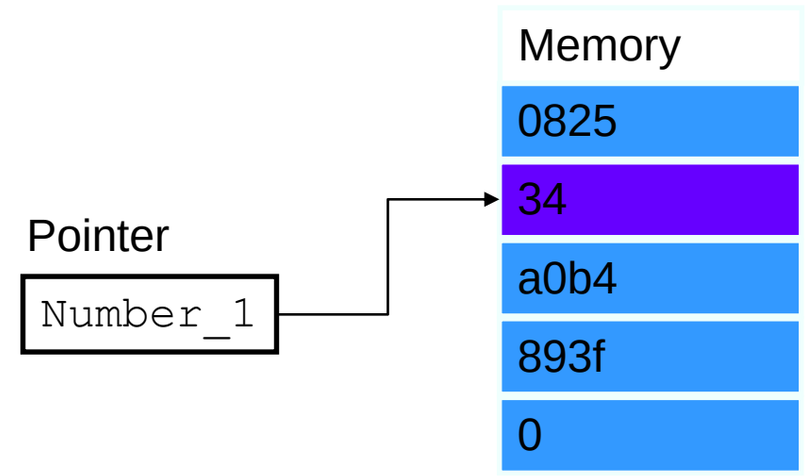
Refresh your memory...

- Before you declare a variable:

- When you declare a variable

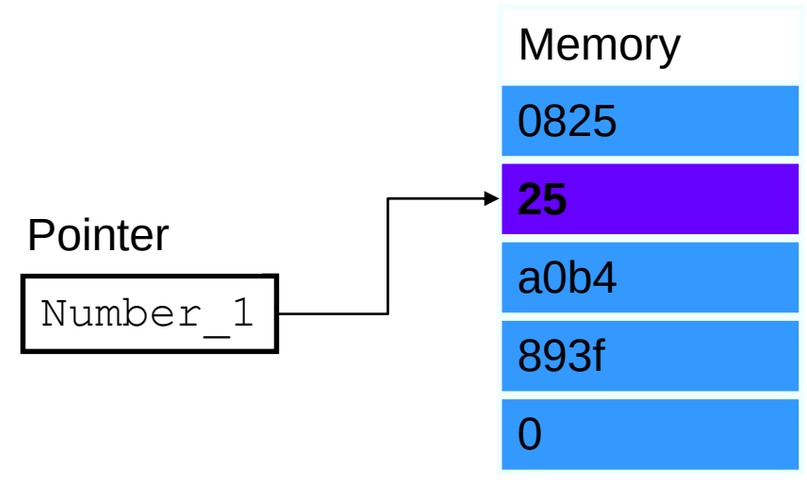
```
int Number_1;
```

A memory block for an integer is allocated
A pointer is created
The pointer is pointing to the memory location allocated



- When you initialize the variable

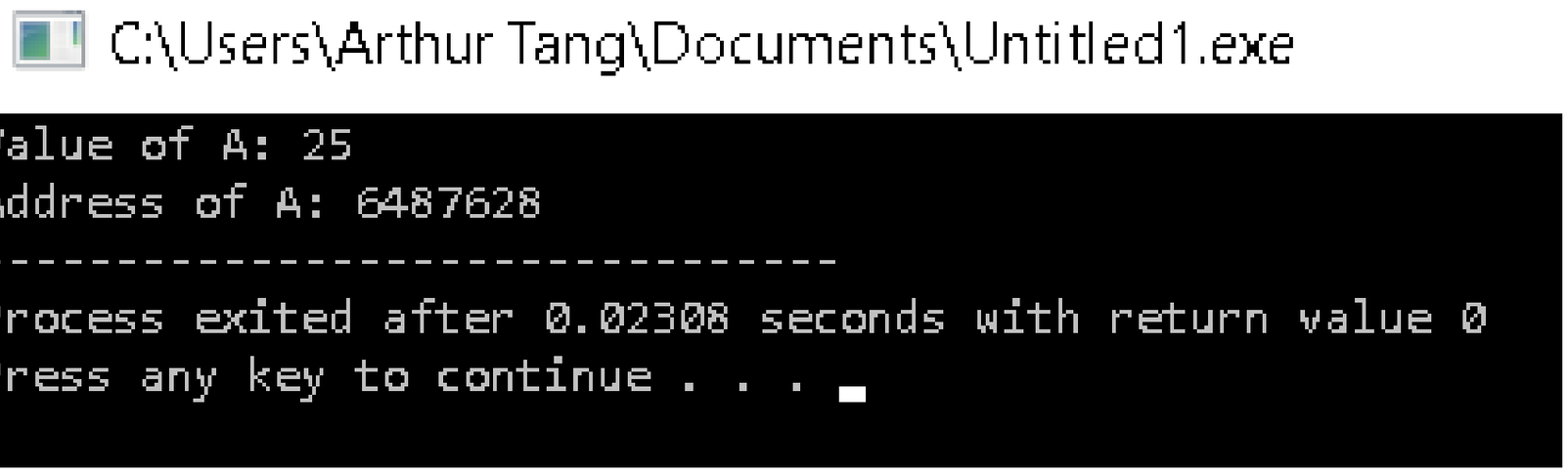
```
int Number_1 = 25;
```



Referencing by Address: **&**, the address operator

- If A is a variable, then &A is the address of the variable, for example:

```
int A = 25;
printf("Value of A: %d\n",A);
printf("Address of A: %d\n",&A);
```



`6487628` is the address of variable A (in decimal value).

- » Byte #6487628 in memory

Variable A is a 32 bit integer (32bit = 4 byte)

- » Byte #6487628, #6487629, #6487630 and #6487631 contains the value of variable A

Refresh your memory...

- The `scanf()` function:

```
int x;
scanf(“%d”,&x);
```

- The `scanf()` function uses the address of a variable as argument
- If you pass the value of a variable as argument:

```
int x=105;
scanf(“%d”,x);
```

- Then the value of the variable is being interpreted as the address of the variable

&, the address operator, another example

```
int A[3] = {1,2,3};
printf("Value of A[0],A[1],A[2]: %d %d %d\n",A[0],A[1],A[2]);
printf("Address of A[0]: %d\n",&A[0]);
printf("Address of A[1]: %d\n",&A[1]);
printf("Address of A[2]: %d\n",&A[2]);
printf("Address of A: %d\n",&A);
```

C:\Users\Arthur Tang\Documents\Untitled1.exe

```
Value of A[0],A[1],A[2]: 1 2 3
Address of A[0]: 6487616
Address of A[1]: 6487620
Address of A[2]: 6487624
Address of A: 6487616
--------------------------------
Process exited after 0.0201 seconds with return value 0
Press any key to continue . . .
```

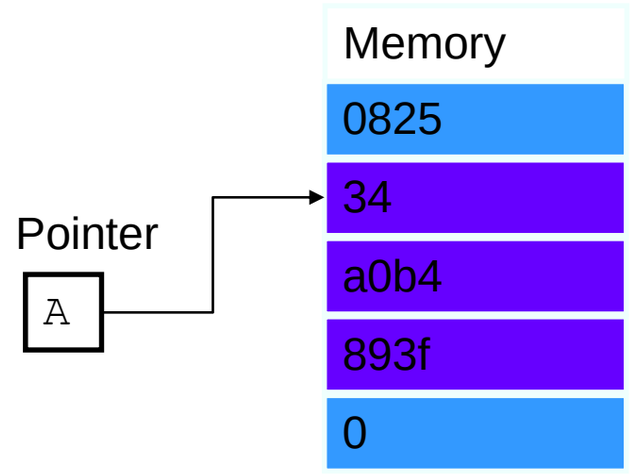
Array and Addresses

- Before you declare a variable:

- When you declare an array

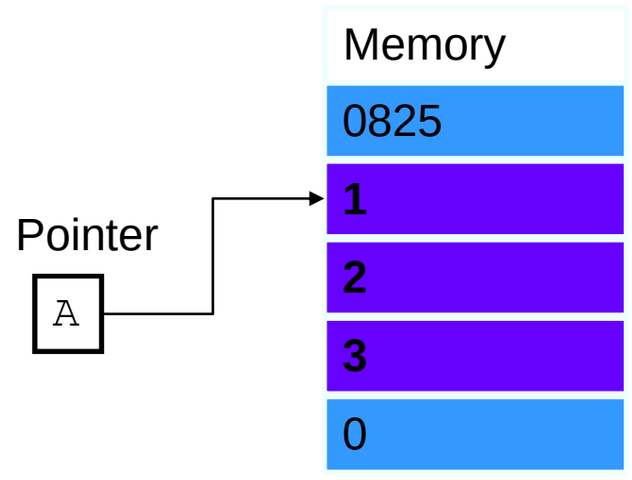
```
int A[3];
```

A continuous memory block for the array is allocated
A pointer is created
The pointer is pointing to the memory location allocated



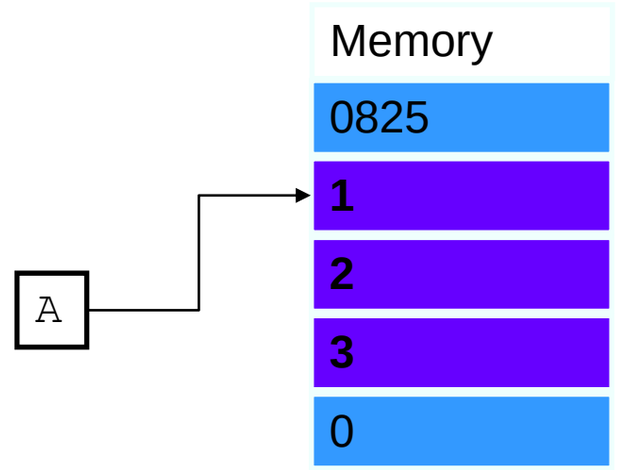
- When you initialize the array

```
A[3] = {1,2,3};
```



Array and Addresses

- The address of `A` is 6487616 (Byte #6487616)
- The address of `A[0]` is 6487616 (Byte #6487616)
  - The address of `A` + 0 x size of int
- The address of `A[1]` is 6487620 (Byte #6487620)
  - The address of `A` + 1 x size of int
  - Size of int is 32 bit (4 byte)
  - 6487616 + 32 bit = 6487616 + 4 byte = 6487620
- The address of `A[2]` is 6487620 (Byte #6487620)
  - The address of `A` + 2 x size of int
  - Size of int is 32 bit (4 byte)
  - 6487616 + 2 x 32 bit = 6487616 + 2 x 4 byte = 6487624

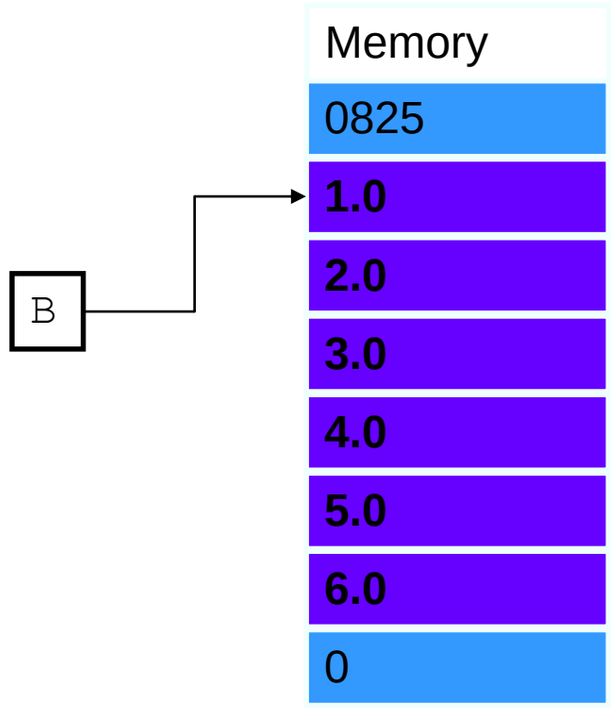


- Note that the address of `A` is the same as the address of `A[0]`

```
C:\Users\Arthur Tang\Documents\Untitled1.exe
Value of A[0],A[1],A[2]: 1 2 3
Address of A[0]: 6487616
Address of A[1]: 6487620
Address of A[2]: 6487624
Address of A: 6487616
--------------------------------
Process exited after 0.0201 seconds with return value 0
Press any key to continue . . .
```

Array and Addresses: Another Example

- **Two dimensional array**
- **float B[2][3] = {{1.0,2.0,3.0},{4.0,5.0,6.0}};**
- The address of `B` is 6487600 (Byte #6487600)
- The address of `B[0][0]` is 6487600 (Byte #6487600)
  - The address of `B` + 0 x size of float + 3 x 0 x size of float
- The address of `B[0][1]` is 6487604 (Byte #6487604)
  - The address of `B` + 1 x size of float + 3 x 0 x size of float
  - Size of float is 32 bit (4 byte)
  - 6487600 + 32 bit = 6487616 + 4 byte = 6487620
- The address of `B[1][2]` is 6487620 (Byte #6487620)
  - The address of `B` + 2 x size of float + 3 x 1 x size of float
  - Size of float is 32 bit (4 byte)
  - 6487600 + 2 x 4 byte + 3 x 1 x 4 byte = 6487600 + 8 + 12 = 6487620




```
C:\Users\Arthur Tang\Documents\Untitled1.exe
Value of B[0][0],B[0][1],B[0][2]: 1.000000 2.000000 3.000000
Value of B[1][0],B[1][1],B[1][2]: 4.000000 5.000000 6.000000
Address of B[0][0]: 6487600
Address of B[0][1]: 6487604
Address of B[0][2]: 6487608
Address of B[1][0]: 6487612
Address of B[1][1]: 6487616
Address of B[1][2]: 6487620
Address of B: 6487600
--------------------------------
Process exited after 0.0131 seconds with return value 0
Press any key to continue . . .
```

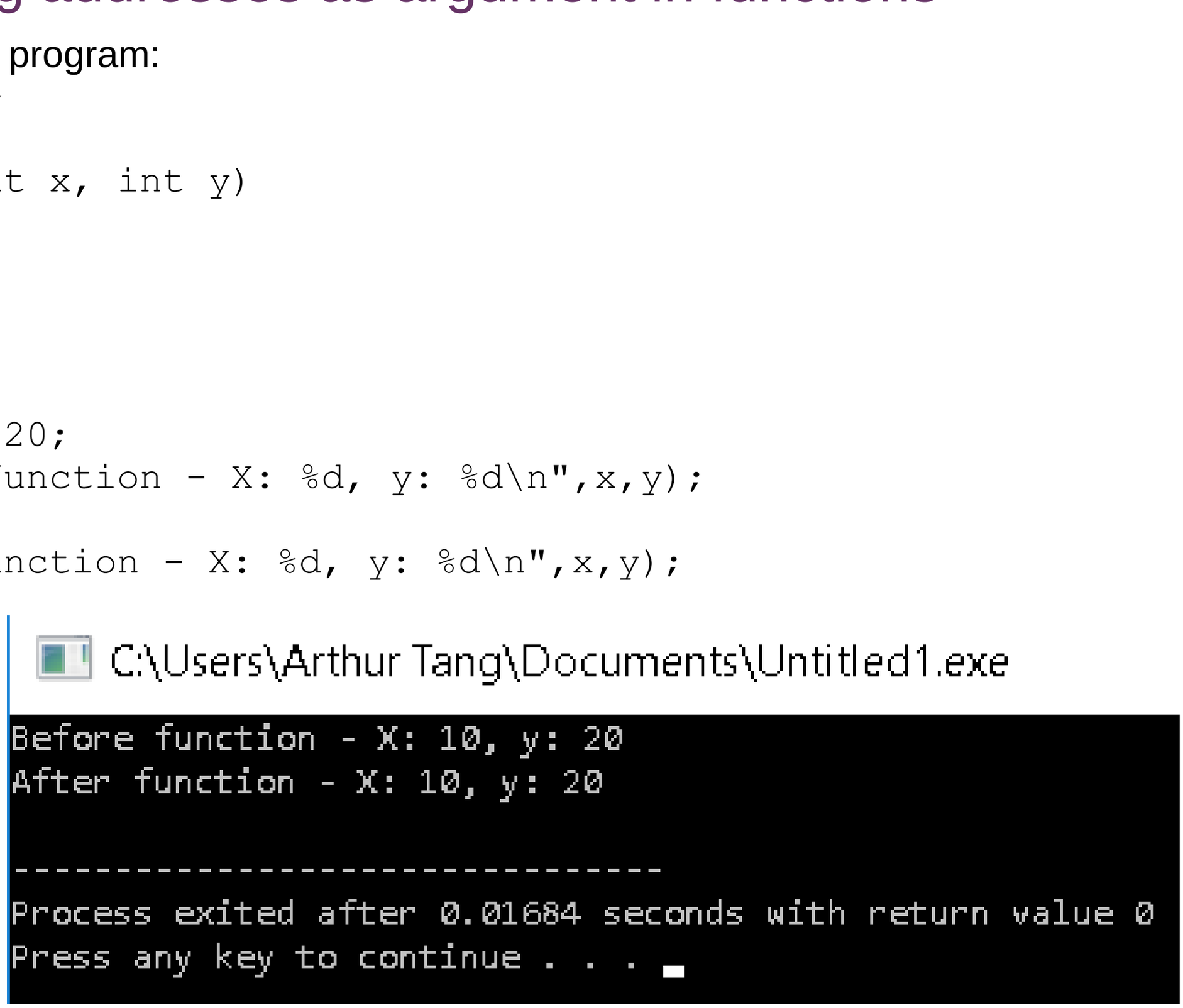
Pass by Reference:
Passing addresses as argument in functions

- Consider the following program:

```
#include <stdio.h>

void MyFunction(int x, int y)
{ x = x+1;
  y = y+1;
}

int main(void)
{ int x = 10, y = 20;
  printf("Before function - X: %d, y: %d\n",x,y);
  MyFunction(x,y);
  printf("After function - X: %d, y: %d\n",x,y);
}
```



What went wrong?

- In the function, parameters x and y were via the function call MyFunction(x, y);
- Then the values of x and y were modified
- When the function returned, x and y were no longer in memory, and a and b retained their original values
- Remember, when you pass by value, the parameter only gets a copy of the corresponding argument; changes to the copy don’t change the original

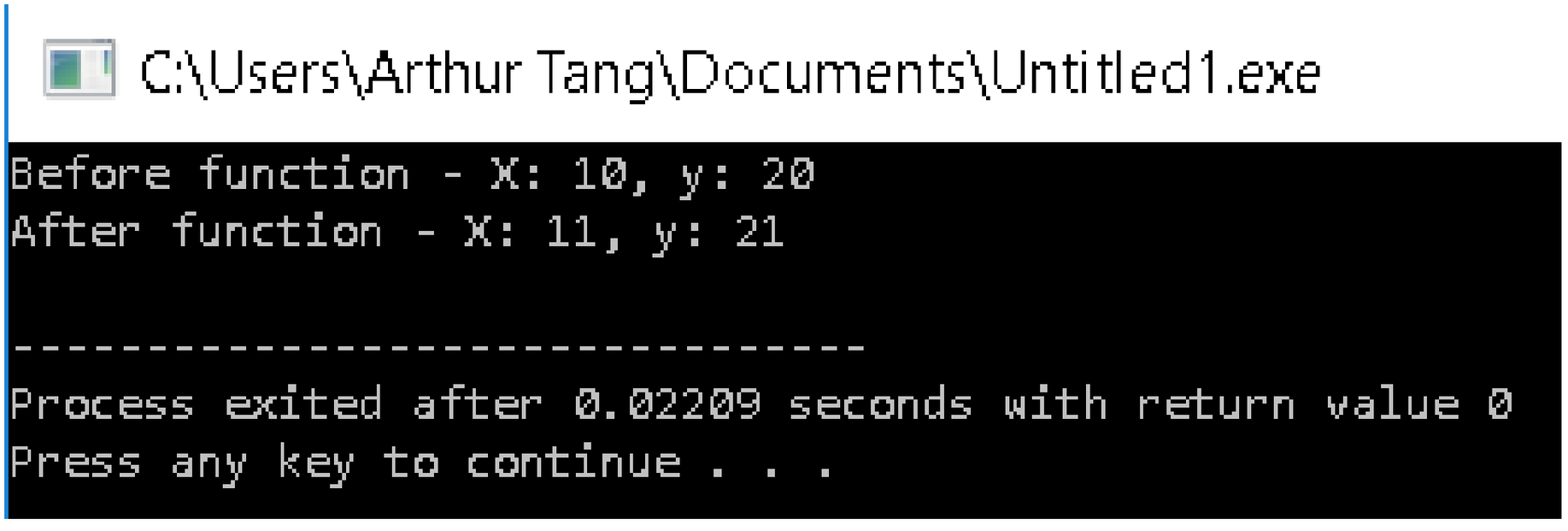
Pass by Reference:
Passing addresses as argument in functions

➢If you change the function definition slightly:

```
#include <stdio.h>

void MyFunction(int &x, int &y)
{ x = x+1;
  y = y+1;
}

int main(void)
{ int x = 10, y = 20;
  printf("Before function - X: %d, y: %d\n",x,y);
  MyFunction(x,y);
  printf("After function - X: %d, y: %d\n",x,y);
}
```



Passing reference parameters

When we pass by reference, the data being passed is the ***address*** of the argument, not the argument itself

The parameter, rather than being a separate variable, is a reference to the same memory that holds the argument – so any change to the parameter is also a change to the argument

Pass by Reference

- The address of the variable (instead of the value of the variable) is passed to the function as argument.
- If the function changes the arguments' value, the changes will be reflected in the program calling it.
- You can use passing by reference to return multiple return values for functions
- You can also pass the entire array (and its size) using passing by reference

Pass by Reference:
Passing array address

➢If you change the function definition slightly:

```
#include <stdio.h>

void Swap(int a[], int size)
{ int tmp = a[0];
  a[0] = a[1];
  a[1] = tmp;
}
```

```
int main(void)
{ int a[2] = {1,5};
  printf("Before function - a[0]: %d, a[1]: %d\n",a[0],a[1]);
  Swap(a,2);
  printf("After function - a[0]: %d, a[1]: %d\n",a[0],a[1]);
}
```

C:\Users\Arthur Tang\Documents\Untitled1.exe

```
Before function - a[0]: 1, a[1]: 5
After function - a[0]: 5, a[1]: 1

--------------------------------
Process exited after 0.01305 seconds with return value 0
Press any key to continue . . .
```

Functions returning multiple values

```
#include <stdio.h>
void stats(float a,float b,float c, float &mean, float &max, float &min)
{ max = min = a;
  float sum = 0;

  if(b > max)   max = b;
  if(b < min)   min = b;
  if(c > max)   max = c;
  if(c < min)   min = c;

  mean = (a+b+c)/3;
}

int main (void)
{ float a = 1, b = 3, c = 8;
  float mean=0 ,max=0, min=0;

  stats(a,b,c,mean,max,min);
  printf("Mean: %f, Max: %f, Min: %f\n",mean,max,min);

  return 0;
}
```

```
C:\Users\Arthur Tang\Documents\Untitled1.exe
Mean: 4.000000, Max: 8.000000, Min: 1.000000

--------------------------------
Process exited after 0.01222 seconds with return value 0
Press any key to continue . . .
```

Passing Entire Arrays to Functions

- Arrays can be passed to functions in their entirety.
- All that is required is
  (1) the address of the first element and
  (2) dimensions of the array.
- The remainder of the array will be passed by reference automatically.

Passing Array's address as argument

```
#include <stdio.h>
void stats(float array[],int size, float &mean, float &max, float &min)
{ max = min = array[0];
  float sum = 0;

  for(int i=0; i<size; i++)
  { if(array[i] > max)   max = array[i];
    if(array[i] < min)   min = array[i];
    sum += array[i];
  }
  mean = sum/size;
}
```

```
int main (void)
{ float A[5] = {1,3,4,6,8};
  float mean=0 ,max=0, min=0;

  stats(A,5,mean,max,min);
  printf("Mean: %f, Max: %f, Min: %f\n",mean,max,min);

  return 0;
}
```

C:\Users\Arthur Tang\Documents\Untitled1.exe

```
Mean: 4.400000, Max: 8.000000, Min: 1.000000


--------------------------------
Process exited after 0.01468 seconds with return value 0
Press any key to continue . . .
```

Passing Array's address as argument

```
#include <stdio.h>
#define SIZE 5
void stats(float array[],int size, float &mean, float &max, float &min)
{ max = min = array[0];
  float sum = 0;

  for(int i=0; i<size; i++)
  { if(array[i] > max)   max = array[i];
    if(array[i] < min)   min = array[i];
    sum += array[i];
  }
  mean = sum/size;
}

int main (void)
{ float A[SIZE] = {1,3,4,6,8};
  float mean=0 ,max=0, min=0;

  stats(A,SIZE,mean,max,min);
  printf("Mean: %f, Max: %f, Min: %f\n",mean,max,min);

  return 0;
}
```

C:\Users\Arthur Tang\Documents\Untitled1.exe

```
Mean: 4.400000, Max: 8.000000, Min: 1.000000


--------------------------------
Process exited after 0.01468 seconds with return value 0
Press any key to continue . . .
```

Note on pass by reference

You can place a space in front of or after the “&”

```
int x;
printf(“Address of X: \n”,&x);
printf(“Address of X: \n”,& x);   // identical

int MyFunction(int & x);
int MyFunction(int &x);           // identical
int MyFunction(int& x);           // identical
```

If you pass arguments by reference, and you don’t want the argument to be modified by the function (e.g. by accident), declare the variable as “const”.

```
int MyFunction(const int Array[])
{   // then any modification of Array[] within the function body
    // will be invalid
}
```

Passing Multidimensional Arrays

How to pass a multidimensional array to a function:

```
void displayBoard(int b[][4]);
// function prototype requires variable name for arrays

void main(){
    int board [4][4];
    ...
    displayBoard(board);
    ...
}
void displayBoard(int b[][4]){
// could also be:   void displayBoard(int b[4][4]){
// but NOT:  void displayBoard(int b[][]){
    ...
}
```

When passing a multidimensional array, only the size of the 1st dimension is optional, the 2nd, 3rd, etc. dimensions must be specified.

Q&A?