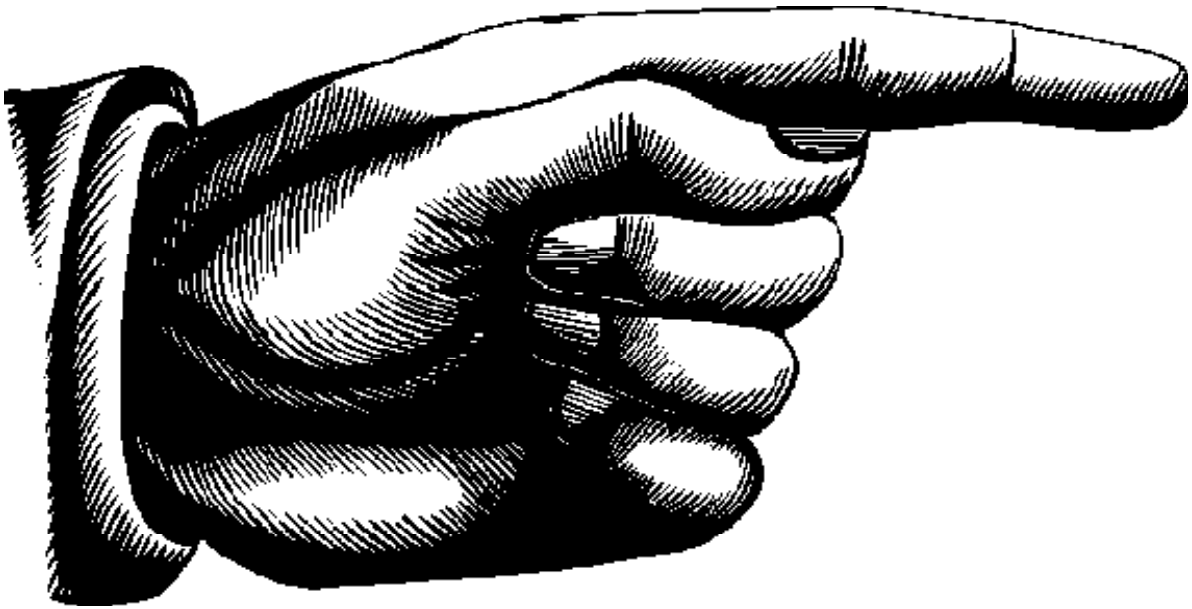
Pointers and Reference parameters

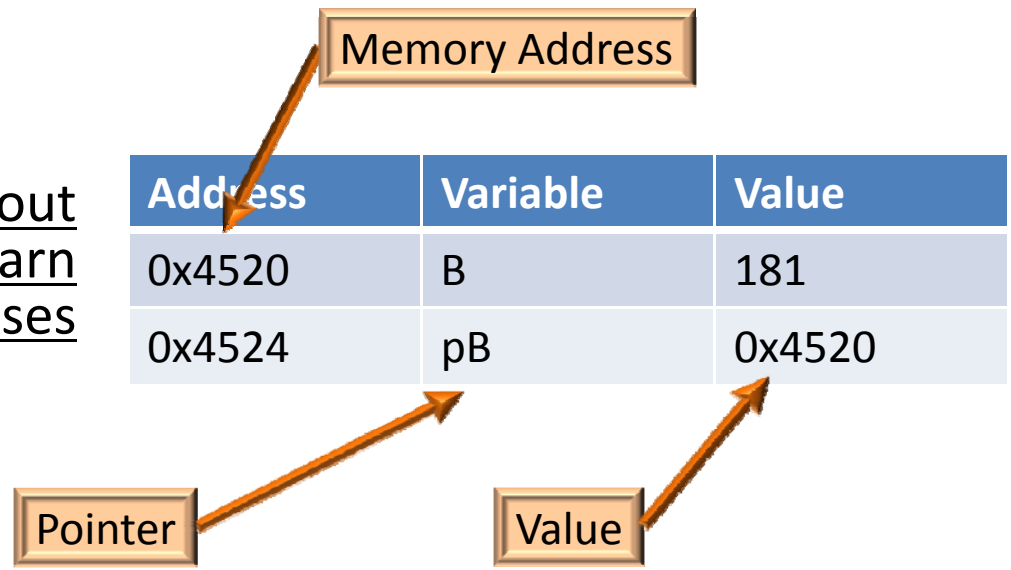


Today's Lecture

Address	Variable	Value
0x4520	B	181
0x4524	pB	0x4520

Pointers are variables that store memory addresses

Before we learn about pointers, we must learn more about addresses



Computer Memory

<u>Memory is just a long list of numbers one after the other</u>

My laptop has over 4 Billion of these numbers

<u>Each number is 8 bits (BYTE)</u>

We combine them to make integers and floating point values

```
0x0039206E  6c 00 00 00 00 00 00 cd
0x0039208C  00 00 00 00 00 00 00 00
0x003920AA  6c 55 67 26 05 8e 10 fb
0x003920C8  10 fb 6c 55 ee 15 05 44
0x003920E6  05 56 10 fb 6c 55 f6 30
```

Computer Memory

Memory 1

Address: 0x00582104 Columns: Auto

```
0x00582104  15 00 00 00 9a 99 91 41 cd cd cd cd 66 66 66 66  ....š™'AÍÍÍÍffff
0x00582114  66 d6 58 40 54 68 69 73 20 69 73 20 61 20 73 74  fÖX@This is a st
0x00582124  72 69 6e 67 00 fd fd fd ab ab ab ab ab ab ab ab  ring.ýýý««««««««
0x00582134  ee fe ee fe 00 00 00 00 00 00 00 00 39 0a 72 63  îþîþ........9.rc
0x00582144  29 90 01 18 00 00 00 00 00 00 00 00 00 00 00 00  )...............
0x00582154  bc ba dc fe 44 00 00 00 03 00 00 00 00 00 00 00  .ºÜþD...........
0x00582164  fd fd fd fd 00 00 00 00 4c 22 58 00 00 00 00 00  ýýýý....L"X.....
0x00582174  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0x00582184  00 00 00 00 00 00 00 00 00 00 00 00 40 22 58 00  ............@"X.
0x00582194  10 22 58 00 00 00 00 00 34 22 58 00 00 00 00 00  ."X.....4"X.....
0x005821A4  28 22 58 00 ec 21 58 00 fd fd fd fd ab ab ab ab  ("X.ì!X.ýýýý««««
0x005821B4  ab ab ab ab 00 00 00 00 00 00 00 00 3e 0a 72 64  ««««........>.rd
0x005821C4  19 94 01 18 00 00 00 00 00 00 00 00 00 00 00 00  .”..............
```

Autos | Locals | Memory 1 | Modules | Watch 1 | Find Symbol Results

int (21)

float (18.2)

```
0x00582104  15 00 00 00 9a 99 91 41 cd cd cd cd 66 66 66 66
0x00582114  66 d6 58 40 54 68 69 73 20 69 73 20 61 20 73 74
0x00582124  72 69 6e 67 00 fd fd fd ab ab ab ab ab ab ab ab
0x00582134  ee fe ee fe 00 00 00 00 00 00 00 00 39 0a 72 63
```

Memory Addresses

Memory addresses in computers are often 32 bits (or nowadays, 64-bits) long, e.g.

01111111111111111111101010001100

Another way to represent an address is to use hexadecimal:

0x 7ffffa8c

Hexadecimal (Base-16)

I have included this chart on your worksheet so you can refer to it.

0000 = 0	1000 = 8
0001 = 1	1001 = 9
0010 = 2	1010 = 10 = a
0011 = 3	1011 = 11 = b
0100 = 4	1100 = 12 = c
0101 = 5	1101 = 13 = d
0110 = 6	1110 = 14 = e
0111 = 7	1111 = 15 = f

Addresses

32-bit address (Binary):

0111 1111 1111 1111 1111 1010 1000 1100

7 f f f f a 8 c

32-bit address (Hex): 0x 7 f f f f a 8 c

Notes:

- In C “0x” indicates a Hexadecimal number
- Convert every four bits to a hex digit

Arithmetic (in Hex)

```
Sum:    0x 90
      + 0x 04
      -------
        0x 94
```

0000 = 0	1000 = 8
0001 = 1	1001 = 9
0010 = 2	1010 = 10 = a
0011 = 3	1011 = 11 = b
0100 = 4	1100 = 12 = c
0101 = 5	1101 = 13 = d
0110 = 6	1110 = 14 = e
0111 = 7	1111 = 15 = f

Arithmetic (in Hex)

Sum with carry:

```
   0x 8c
 + 0x 04
 --------
   0x ??
```

0000 = 0	1000 = 8
0001 = 1	1001 = 9
0010 = 2	1010 = 10 = a
0011 = 3	1011 = 11 = b
0100 = 4	1100 = 12 = c
0101 = 5	1101 = 13 = d
0110 = 6	1110 = 14 = e
0111 = 7	1111 = 15 = f

Arithmetic (in Hex)

Sum with carry:

$$\begin{array}{r} 0x\ 8c \\ +\ 0x\ 04 \\ \hline 0x\ ?? \end{array}$$

- What is “c + 4”?
- In decimal it is “12 + 4 = 16” which is Hex “10” (0 and carry 1)

Arithmetic (in Hex)

Sum with carry:

```
      1
   0x 8c
 + 0x 04
 --------
   0x ??
```

- What is “c + 4”?
- In decimal it is “12 + 4 = 16” which is Hex “10” (0 and carry 1)

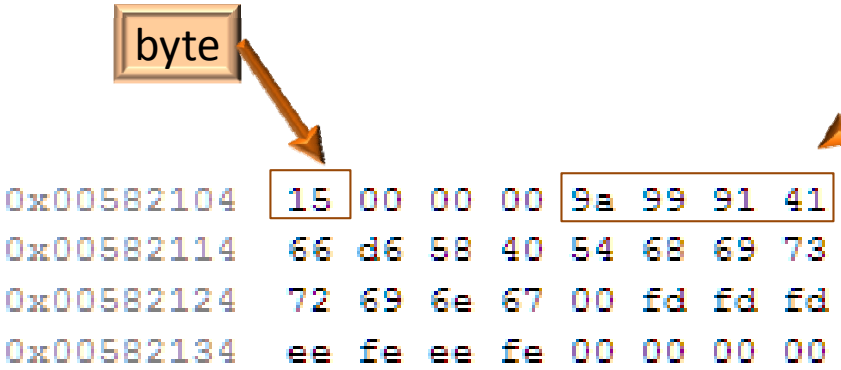
Arithmetic (in Hex)

Sum with carry:

```
      1
   0x 8c
 + 0x 04
 --------
   0x 90
```

- What is “c + 4”?
- In decimal it is “12 + 4 = 16” which is Hex “10” (0 and carry 1)

Bytes and Words

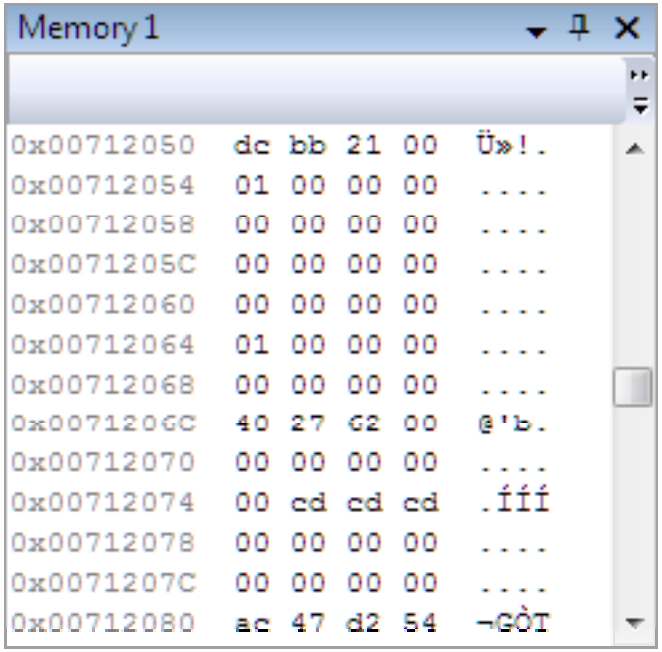


Remember

8 bits = 1 byte

32 bits = 4 bytes = 1 word.

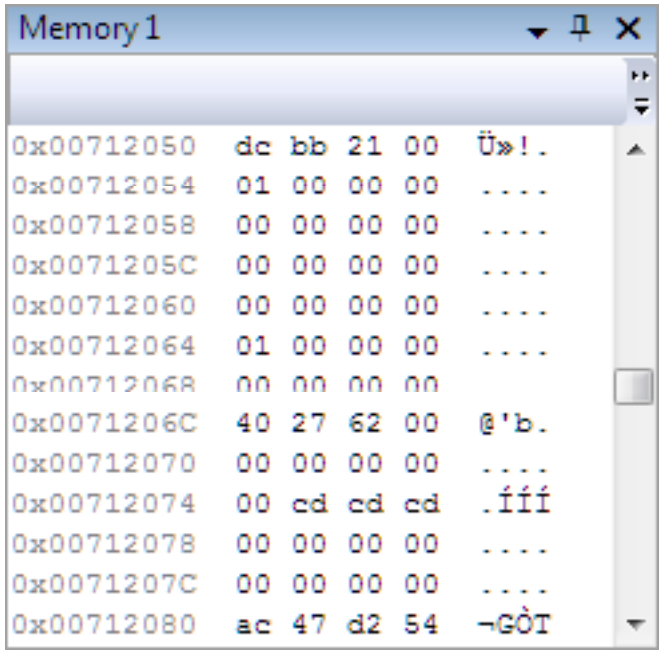
32-bit address machines are addressed by bytes so consecutive words have addresses that differ by four



32-bit Addresses

Here are three consecutive 32-bit addresses (in Hex) of words:

0x00712050	dc bb 21 00
0x00712054	01 00 00 00
0x00712058	00 00 00 00



Pointers

Pointers are variables that contain addresses

Just like other variables, they must be declared before being used

Declaration:

```
int *p;     /* instead of int p for integers */
```

int * means p is a pointer variable that stores the address of an integer variable

Pointer Initialization

Declaration:

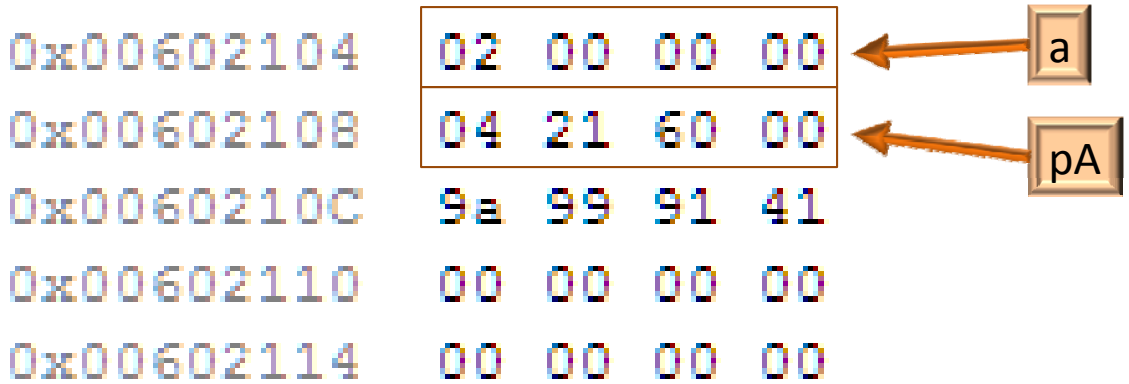
```
int a = 2;    /* a is an integer */
int *pA = &a; /* pA is a pointer containing
                 the address of a */
```

"&" operator means "address of"

Read it as "at"

The Address Game

```
int a = 2;
int *pA = &a;
```



An Intel processor is called a little endian processor because it stores values with the least significant byte first. You read it in reverse order.

0x00602104 in memory will be: 04 21 60 00

Example Program

"%x" prints the hexadecimal value

Operators:

& "address of"

* "dereference"

```
int a = 21;
int *pA = &a;


printf("%d\n", a);
printf("%x\n", a);
printf("%x\n", &a);
printf("%x\n", pA);
printf("%d\n", *pA);
printf("%x\n", &pA);
```

Output

```
21
15
bfee861c
bfee861c
21
bfee8618
```

```
#include<stdio.h>

int main()
{
        int a = 15, b = 38;
        int *c = &a;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);

        a = 49;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);

        c = &b;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);
}
```

```
#include<stdio.h>

int main()
{
        int a = 15, b = 38;
        int *c = &a;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);

        a = 49;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);

        c = &b;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);
}
```

First Section

Declares a and b as integers
Declares c as a pointer that contains the *address* of a ("points to a")

```
int a = 15, b = 38;
int *c = &a;
printf("%x : %d\n", &a, a);
printf("%x : %d\n", &b ,b);
printf("%x : %x : %d\n", &c, c, *c);
```

Address	Memory	Name
0xeffffa94	15	a
0xeffffa90	38	b
0xeffffa8c	**0xeffffa94**	c

First Section

Declares a and b as integers
Declares c as a pointer that contains the *address* of a ("points to a")

```
int a = 15, b = 38;
int *c = &a;
printf("%x : %d\n", &a, a);
printf("%x : %d\n", &b ,b);
printf("%x : %x : %d\n", &c, c, *c);
```

Address	Memory	Name
0xeffffa94	15	a
0xeffffa90	38	b
0xeffffa8c	**0xeffffa94**	c

Output:
effffa94 15
effffa90 38
effffa8c effffa94 15

First Section

Declares a and b as integers
Declares c as a pointer that contains the *address* of a ("points to a")

```
int a = 15, b = 38;
int *c = &a;
printf("%x : %d\n", &a, a);
printf("%x : %d\n", &b ,b);
printf("%x : %x : %d\n", &c, c, *c);
```

Note the difference between *C in variable declaration and *C in printf

Output:
effffa94 15
effffa90 38
effffa8c effffa94 15

```c
#include<stdio.h>

int main()
{
        int a = 15, b = 38;
        int *c = &a;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);

        a = 49;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);

        c = &b;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);
}
```

Second Section

```
a = 49;
printf("%x : %d\n", &a, a);
printf("%x : %d\n", &b ,b);
printf("%x : %x : %d\n", &c, c, *c);
```

Address	Memory	Name
0xeffffa94	~~15~~ **49**	A
0xeffffa90	38	B
0xeffffa8c	**0xeffffa94**	C

Second Example

```
a = 49;
printf("%x : %d\n", &a, a);
printf("%x : %d\n", &b ,b);
printf("%x : %x : %d\n", &c, c, *c);
```

Address	Memory	Name
0xeffffa94	~~15~~ **49**	A
0xeffffa90	38	B
0xeffffa8c	**0xeffffa94**	C

Output:

effffa94 49

effffa90 38

effffa8c effffa94 49

```c
#include<stdio.h>

int main()
{
        int a = 15, b = 38;
        int *c = &a;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);

        a = 49;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);

        c = &b;
        printf("%x : %d\n", &a, a);
        printf("%x : %d\n", &b ,b);
        printf("%x : %x : %d\n", &c, c, *c);
}
```

Third Section

```
c = &b;  /* c now points to b */
printf("%x : %d\n", &a, a);
printf("%x : %d\n", &b ,b);
printf("%x : %x : %d\n", &c, c, *c);
```

Address	Memory	Name
0xeffffa94	49	A
0xeffffa90	38	B
0xeffffa8c	**0xeffffa90**	C

Third Section

```
c = &b;  /* c now points to b */
printf("%x : %d\n", &a, a);
printf("%x : %d\n", &b ,b);
printf("%x : %x : %d\n", &c, c, *c);
```

Address	Memory	Name
0xeffffa94	49	A
0xeffffa90	38	B
0xeffffa8c	**0xeffffa90**	C

Output:
effffa94 49
effffa90 38
effffa8c effffa90 38

Reference parameters

A valuable use for pointers:

Passing addresses to a function

Argument & Returned Value

Consider a function call y=f(x).

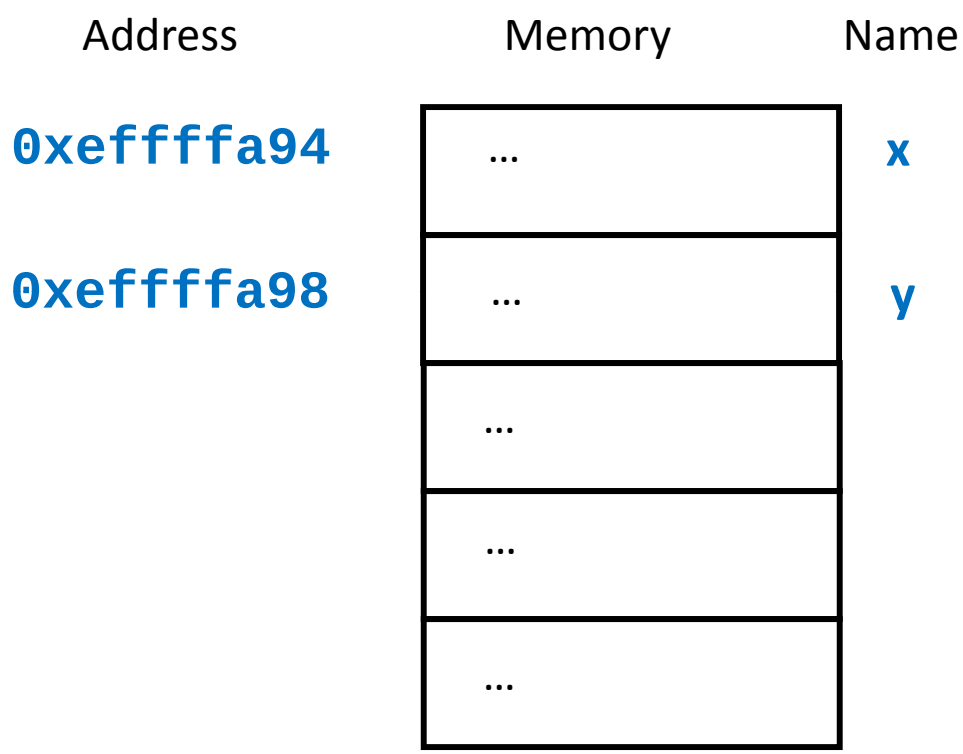
- The value x is passed to the function f
- A value is returned and assigned to y.

- By *passed* we mean that the value of argument x is *copied* to the parameter in the function. Some calculation is performed and the result is returned and assigned to y.

Example

```
int x, y;
x = 5;
y = Square(x);

int Square(int t)
{
  return t*t
}
```



Example

```
int x, y;
x = 5;
y = Square(x);

int Square(int t)
{
  return t*t
}
```

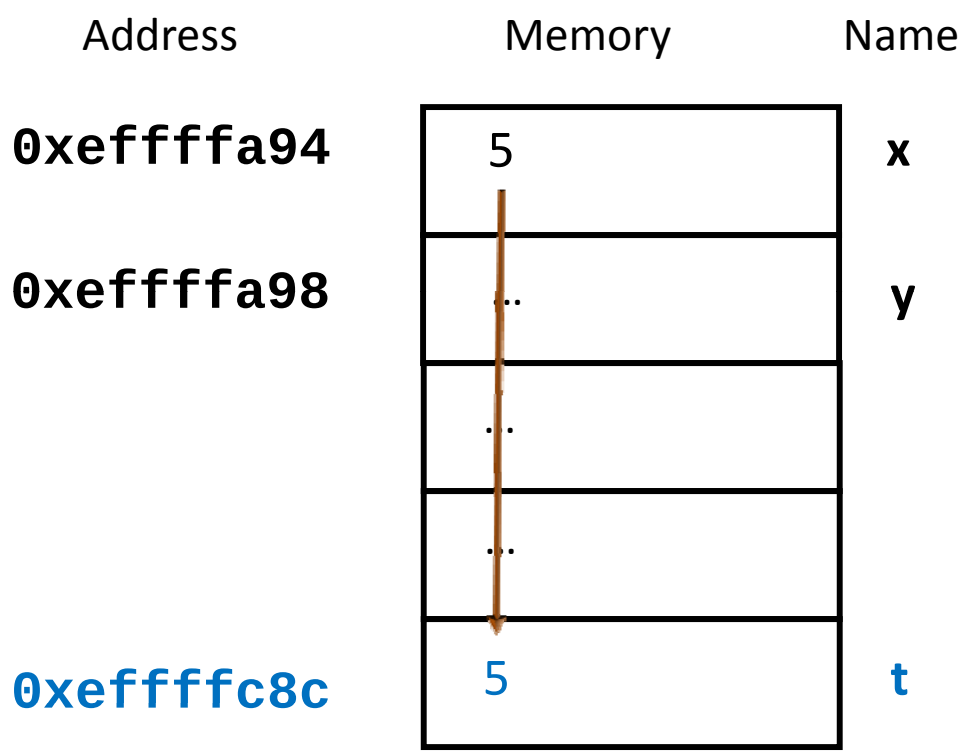
Address	Memory	Name
0xeffffa94	5	x
0xeffffa98	…	y
	…	
	…	
	…	

Example

```
int x, y;
x = 5;
y = Square(x);

int Square(int t)
{
  return t*t
}
```

The call Square(x) :
creates a variable t
copies the value of x to t



Example

```
int x, y;
x = 5;
y = Square(x);

int Square(int t)
{
  return t*t
}
```

The call Square(x) :

- **creates** a variable t
- **copies** the value of x to t
- **calculates** t * t
- **returns** t

Address	Memory	Name
0xeffffa94	5	x
0xeffffa98	…	y
	…	
	…	
0xeffffc8c	5	t
0xeffffc90	25	temp

y=f(x)

Only *one* valued returned

What if we want to return more than one value?

- Solution is to use pointers to variables in the calling function

How to do this in C

The approach is to pass the *address (using the & operator)* of the value to be modified.

We call such a parameter a *reference* parameter.

Use the * operator to change the reference parameter value

Function Reference Params

```
int val = 10;
MyFun(&val);
printf("%d",val);
```

Name	Address	Value
val	0xeffffa90	10

```
void MyFun(int *param)
{
    *param = 27;
}
```

Function Reference Params

```
int val = 10;
MyFun(&val);
printf("%d",val);
```

Name	Address	Value
val	0xeffffa90	10

Name	Address	Value
param	0xefffea88	0xefffa90

```
void MyFun(int *param)
{
    *param = 27;
}
```

Function Reference Params

```
int val = 10;
MyFun(&val);
printf("%d",val);
```

Name	Address	Value
val	0xeffffa90	27

Name	Address	Value
param	0xefffea88	0xefffa90

```
void MyFun(int *param)
{
    *param = 27;
}
```

Function Reference Params

```
int val = 10;
MyFun(&val);
printf("%d",val);
```

Prints: 27

Name	Address	Value
val	0xeffffa90	27

The memory used by the function is destroyed when it returns.

```
void MyFun(int *param)
{
    *param = 27;
}
```

What will this do different?

```
int val = 10;
MyFun2(val);
printf("%d",val);
```

Name	Address	Value
val	0xeffffa90	10

```
void MyFun2(int param)
{
    param = 27;
}
```

Cards program

```
/* Create a random card and suit */
/* This will compute a random
suit1 = rand() % 4;

/* This will compute a random
card1 = rand() % 13 + 1;

do
{
    /* Create a random card and suit */
    /* This will compute a random number from 0 to 3 */
    suit2 = rand() % 4;

    /* This will compute a random number from 1 to 13 */
    card2 = rand() % 13 + 1;

} while(card1 == card2 && suit1 == suit2);
```

This program repeats code. We don't like to do that. But, we could not put the card draw into a function because a function can only return one value, or so we thought!

Solution, pass by reference using pointers

```
/* Create a random card and suit */
DrawCard(&card1, &suit1);

do
{
    DrawCard(&card2, &suit2);
} while(card1 == card2 && suit1 == suit2);
```

Don't forget:

*suit ← to set the value
&card1 ← to get the address

Pass with &
Set with *

```
void DrawCard(int *card, int *suit)
{
    /* This will compute a
       random number from 0 to 3 */
    *suit = rand() % 4;

    /* This will compute a random
       number from 1 to 13 */
    *card = rand() % 13 + 1;
}
```