Pointers and Reference parameters
Today’s Lecture

Pointers are variables that store memory addresses

<table>
<thead>
<tr>
<th>Address</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4520</td>
<td>B</td>
<td>181</td>
</tr>
<tr>
<td>0x4524</td>
<td>pB</td>
<td>0x4520</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x4520</td>
<td>B</td>
<td>181</td>
</tr>
<tr>
<td>0x4524</td>
<td>pB</td>
<td>0x4520</td>
</tr>
</tbody>
</table>
Computer Memory

Memory is just a long list of numbers one after the other
My laptop has over 4 Billion of these numbers

Each number is 8 bits (BYTE)
We combine them to make integers and floating point values
Computer Memory

int (21)

float (18.2)
Memory Addresses

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

\[ 0111111111111111111110101001100 \]

Another way to represent an address is to use hexadecimal:

\[ 0x \text{7fff}f\text{fa}8c \]
Hexadecimal (Base-16)

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

<table>
<thead>
<tr>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>a</td>
</tr>
<tr>
<td>1011</td>
<td>b</td>
</tr>
<tr>
<td>1100</td>
<td>c</td>
</tr>
<tr>
<td>1101</td>
<td>d</td>
</tr>
<tr>
<td>1110</td>
<td>e</td>
</tr>
<tr>
<td>1111</td>
<td>f</td>
</tr>
</tbody>
</table>
Addresses

32-bit address (Binary):
0111 1111 1111 1111 1111 1010 1000 1100
 7 f f f f f a 8 c

32-bit address (Hex): 0x 7 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: \(0x\ 90\)
+ \(0x\ 04\)
\(0x\ 94\)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>10 = a</td>
</tr>
<tr>
<td>1011</td>
<td>11 = b</td>
</tr>
<tr>
<td>1100</td>
<td>12 = c</td>
</tr>
<tr>
<td>1101</td>
<td>13 = d</td>
</tr>
<tr>
<td>1110</td>
<td>14 = e</td>
</tr>
<tr>
<td>1111</td>
<td>15 = f</td>
</tr>
</tbody>
</table>
Arithmetic (in Hex)

Sum with carry: \[ 0x \text{ 8c} \]
\[ + \text{ 0x 04} \]
\[ \text{0x ??} \]

<table>
<thead>
<tr>
<th>Hex</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>a</td>
</tr>
<tr>
<td>1011</td>
<td>b</td>
</tr>
<tr>
<td>1100</td>
<td>c</td>
</tr>
<tr>
<td>1101</td>
<td>d</td>
</tr>
<tr>
<td>1110</td>
<td>e</td>
</tr>
<tr>
<td>1111</td>
<td>f</td>
</tr>
</tbody>
</table>
Arithmetic (in Hex)

Sum with carry:

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:

\[ 0x \ 8c + 0x \ 04 = 0x \ ?? \]

- 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:

\[
\begin{array}{c}
1 \\
0x \ 8c \\
+ \ 0x \ 04 \\
\hline
0x \ 90
\end{array}
\]

- 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:

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00712050</td>
<td>dc bb 21 00</td>
</tr>
<tr>
<td>0x00712054</td>
<td>01 00 00 00</td>
</tr>
<tr>
<td>0x00712058</td>
<td>00 00 00 00</td>
</tr>
</tbody>
</table>

![Memory View](image)
Pointers

*Pointers* are variables that contain addresses

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

Declaration:

```c
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

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00602104</td>
<td>02 00 00 00</td>
<td></td>
</tr>
<tr>
<td>0x00602108</td>
<td>04 21 60 00</td>
<td></td>
</tr>
<tr>
<td>0x0060210C</td>
<td>9a 99 91 41</td>
<td></td>
</tr>
<tr>
<td>0x00602110</td>
<td>00 00 00 00</td>
<td></td>
</tr>
<tr>
<td>0x00602114</td>
<td>00 00 00 00</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

"%x" prints the hexadecimal value

Operators:
&  "address of"
*  "dereference"

```
Output
21
15
bfee861c
bfee861c
21
bfee8618
```
#include<stdio.h>

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

```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”)

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xefffffff94</td>
<td>15</td>
<td>a</td>
</tr>
<tr>
<td>0xefffffff90</td>
<td>38</td>
<td>b</td>
</tr>
<tr>
<td>0xefffffff8c</td>
<td>0xefffffff94</td>
<td>c</td>
</tr>
</tbody>
</table>

```c
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);
```
First Section

Declarations:

- 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);
```

**Output:**

efff9a94 15
efff9a90 38
efff8c efff9a94 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);
```

Output:
effffa94 15
effffa90 38
effffa8c effffa94 15
#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);
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xefffffff94</td>
<td>15 49</td>
<td>A</td>
</tr>
<tr>
<td>0xefffffff90</td>
<td>38</td>
<td>B</td>
</tr>
<tr>
<td>0xefffffff8c</td>
<td></td>
<td>C</td>
</tr>
</tbody>
</table>
Second Example

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

Output:
efffa94  49
efffa90  38
efffa8c  effmpeg9a4  49

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xefffa94</td>
<td>15 49</td>
<td>A</td>
</tr>
<tr>
<td>0xefffa90</td>
<td>38</td>
<td>B</td>
</tr>
<tr>
<td>0xefffa8c</td>
<td></td>
<td>C</td>
</tr>
</tbody>
</table>
```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);

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xefffffa94</td>
<td>49</td>
<td>A</td>
</tr>
<tr>
<td>0xefffffa90</td>
<td>38</td>
<td>B</td>
</tr>
<tr>
<td>0xefffffa8c</td>
<td></td>
<td>C</td>
</tr>
</tbody>
</table>
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);

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

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

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

```
Address           Memory           Name
0xefffffa94       ...               x
0xefffffa98       ...               y
...               ...
...               ...
...               ...
...               ...
```
Example

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

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

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xefffff9a94</td>
<td>5</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>y</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Example

```c
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`

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xefffffa94</td>
<td>5</td>
<td>x</td>
</tr>
<tr>
<td>0xefffffa98</td>
<td>...</td>
<td>y</td>
</tr>
<tr>
<td>0xefffffc8c</td>
<td>5</td>
<td>t</td>
</tr>
</tbody>
</table>
Example

```c
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`
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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>0xefffa90</td>
<td>10</td>
</tr>
</tbody>
</table>

```c
void MyFun(int *param)
{
    *param = 27;
}
```
```c
int val = 10;
MyFun(&val);
printf("%d", val);
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>0xeffffffa90</td>
<td>10</td>
</tr>
</tbody>
</table>

```c
void MyFun(int *param)
{
    *param = 27;
}
```
Function Reference Params

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

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>0xefff9a0</td>
<td>27</td>
</tr>
</tbody>
</table>
```

```c
void MyFun(int *param)
{
    *param = 27;
}
```
Function Reference Params

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

Prints: 27

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>0xefffa90</td>
<td>27</td>
</tr>
</tbody>
</table>

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

```c
void MyFun(int *param)
{
    *param = 27;
}
```
What will this do different?

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>val</td>
<td>0xefffffa90</td>
<td>10</td>
</tr>
</tbody>
</table>

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

/* Create a random card and suit */
/* This will compute a random number from 0 to 3 */
suit1 = rand() % 4;

/* This will compute a random number from 1 to 13 */
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

```c
/* 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 *

```c
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;
}
```