

# CSCI 390 – Special Topics in C++

## Lecture 10

9/20/18

Time To Turn Off Cell Phones

# **if** Statement

- Syntax:

**if** (<test exp> ) <true statement>;

or, more typically:

**if** (<test exp> )

{

    <true statement>;

    ...

}

# if else Statement

- Syntax:

**if** (<test exp>) <true stmt>; **else** <false stmt>;

or, more typically:

**if** (<test exp>)

{

    <true stmt>;

    ...

}

**else**

{

    <false stmt>;

    ...

}

# if else Statement (cont)

## Example

```
#include <iostream>

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        std::cout << "Only one arg!" << std::endl;
    }
    else
    {
        std::cout << "Lots of args!" << std::endl;
    }

    return 0;
}
```

Only one arg!

...Program finished with exit code 0  
Press ENTER to exit console.

# if else if Statement (cont)

## Example

```
#include <iostream>

int main(int argc, char *argv[])
{
    if (argc == 1)
    {
        std::cout << "Only one arg!" << std::endl;
    }
    else if (argc == 2)
    {
        std::cout << "2 args!" << std::endl;
    }
    else
    {
        std::cout << "Lots of args!" << std::endl;
    }

    return 0;
}
```

Only one arg!

...Program finished with exit code 0  
Press ENTER to exit console.

# if else Statement (cont)

## Comparing **signed** and **unsigned**

- Comparing **signed** and **unsigned** is dangerous.
- The time has come to understand why.

```
#include <iostream>
int main()
{
    auto lhs{0};
    auto rhs{1u};
    if (lhs < rhs)
        std::cout << "lhs < rhs" << std::endl;
    else
        std::cout << "lhs >= rhs" << std::endl;

    // lhs now -1, and -1 < 1. Right?
    --lhs;

    if (lhs < rhs)
        std::cout << "lhs < rhs" << std::endl;
    else
        std::cout << "lhs >= rhs" << std::endl;

    return 0;
}
```

```
lhs < rhs
lhs >= rhs
```

...Program finished with exit code 0  
Press ENTER to exit console.

# WHY?

# if else Statement (cont)

## Comparing **signed** and **unsigned**

- Comparing **signed** and **unsigned** is dangerous.
- The time has come to understand why.

```
#include <iostream>
int main()
{
    auto lhs{0};
    auto rhs{1u};
    if (lhs < rhs)
        std::cout << "lhs < rhs" << std::endl;
    else
        std::cout << "lhs >= rhs" << std::endl;

    // lhs now -1, and -1 < 1. Right?
    --lhs;
    std::cout << "lhs: " << lhs << ", rhs: " <<
        rhs << std::endl;

    if (lhs < rhs)
        std::cout << "lhs < rhs" << std::endl;
    else
        std::cout << "lhs >= rhs" << std::endl;

    return 0;
}
```

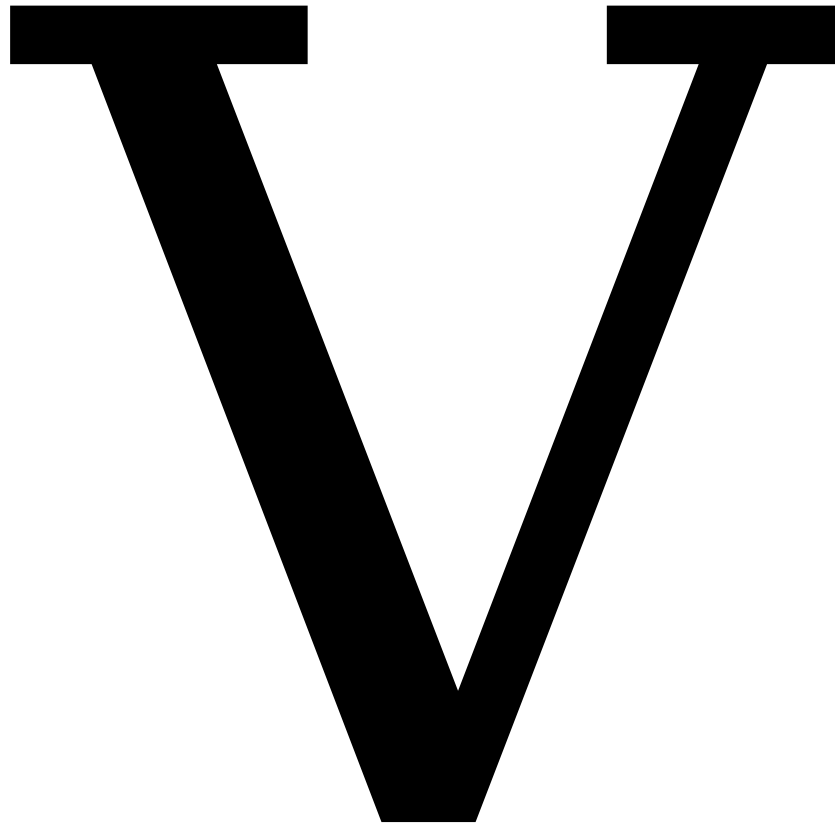
```
lhs < rhs
lhs: -1, rhs: 1
lhs >= rhs
```

...Program finished with exit code 0  
Press ENTER to exit console.

# WHY?

# Step 1

## What is this?





# Step 2

## Nature Follows Math

- If a mathematical theory accurately describes Nature, then Nature follows the Math and Math can be used to predict Nature.
- When General Relativity was finally confirmed, a reporter asked Einstein if he was surprised. He simply responded “The Math is correct.”
- A great philosopher from the 40’s and 50’s gave an unforgettable lecture on this topic.
- Here is a simple example of the idea:
  - If  $x + 1 = 0$ , then, by algebra,  $x = -1$ .

# Step 3

## A Simple 4-Bit ALU Counter

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

# Step 3

## A Simple 4-Bit ALU Counter

Binary	Decimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15
0000	0

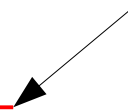
Overflow!

# Step 3

## A Simple 4-Bit ALU Counter

	Binary	Decimal
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
	0111	7
	1000	8
	1001	9
	1010	10
	1011	11
	1100	12
	1101	13
	1110	14
x	1111	15
x+1	0000	0

Overflow!




# Step 3

## A Simple 4-Bit ALU Counter

????	Binary	Decimal
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
	0111	7
	1000	8
	1001	9
	1010	10
	1011	11
	1100	12
	1101	13
x	1110	14
x+1	-1	15
	0000	0

Overflow!



# Step 3

## A Simple 4-Bit ALU Counter

????	Binary	Decimal
	0000	0
	0001	1
	0010	2
	0011	3
	0100	4
	0101	5
	0110	6
	0111	7
	1000	8
	1001	9
	1010	10
	1011	11
	1100	12
	1101	13
x	-2	14
x+1	-1	15
	0000	0

Overflow!

# Step 3

## A Simple 4-Bit ALU Counter

Signed	Binary	Unsigned
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
-8	1000	8
-7	1001	9
-6	1010	10
-5	1011	11
-4	1100	12
-3	1101	13
-2	1110	14
-1	1111	15
0	0000	0

Two's Complement  
Signed Overflow!

Unsigned Overflow!

# Step 3

## A Simple 4-Bit ALU Counter

Signed	Binary	Unsigned
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
-8	1000	8
-7	1001	9
-6	1010	10
-5	1011	11
-4	1100	12
-3	1101	13
-2	1110	14
-1	1111	15
0	0000	0

In general, you can't compare signed and unsigned because they overflow at different places.

Two's Complement  
Signed Overflow!

Unsigned Overflow!



# Step 4

## Putting It All Together

Signed	Binary	Unsigned
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
-8	1000	8
-7	1001	9
-6	1010	10
-5	1011	11
-4	1100	12
-3	1101	13
-2	1110	14
-1	1111	15
0	0000	0

- Here's why **if (lhs < rhs)** did not work as expected.
- The **--lhs** yielded -1.
- **lhs < rhs** can't be done so the compiler converted **lhs** to unsigned, and that gave 15.
- $15 > 1$ , and that is why the test did not work as expected.
- It makes as much sense as:  
 $V + I = W$

# Step 4

## However

Signed	Binary	Unsigned
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
-8	1000	8
-7	1001	9
-6	1010	10
-5	1011	11
-4	1100	12
-3	1101	13
-2	1110	14
-1	1111	15
0	0000	0

If you can guarantee operands are restricted to this range (non-negative), you can compare signed and unsigned.

This is not always easy.

# **break** Statement

- A **break** statement is place inside the body of loop or switch (covered soon) statement.
- A **break** transfers control outside the innermost scope.
- When used inside a loop body, it aborts the loop. It can be used to implement a generalized loop.

# break Statement Example

- Exiting from an infinite loop:

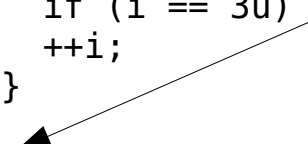
```
#include <iostream>

int main()
{
    auto Sum{0u};
    auto i{1u};

    for ( ; ; )
    {
        Sum += i;
        if (i == 3u) break;
        ++i;
    }

    std::cout << "Sum: " << Sum << std::endl;

    return 0;
}
```



Sum: 6

...Program finished with exit code 0  
Press ENTER to exit console.

# **continue** Statement

- A **continue** statement is place inside the body of loop.
- A **continue** skips the rest of the loop body and immediately begins the next loop cycle.

# continue/break Statements

## Example

```
#include <iostream>

int main()
{
    auto Sum{0u};
    auto i{1u};

    for ( ; ; )
    {
        Sum += i;
        if (i++ < 3u) continue;
        std::cout << "Sum: " << Sum << std::endl;
        break;
    }

    return 0;
}
```

Sum: 6

...Program finished with exit code 0  
Press ENTER to exit console.

# switch Statement

- A **switch** statement transfers control to one of many statements based on the value of an expression.

- Syntax:

```
switch (<expression>)  
{  
    case <constant>:  
        <statement>s;  
    case <constant>:  
        <statement>s;  
    . . .  
    default:  
        <statement>s;  
}
```

# switch Statement

- A **switch** statement is often used to implement:  
**if ( ) else if ( ) else if( ) ... else ;**
- Cannot declare variables inside { }.



# switch Statement Example

```
#include <iostream>

int main(int argc, char *argv[])
{
    switch (argc)
    {
        case 1:
            std::cout << "1 arg" << std::endl;
        case 2:
            std::cout << "2 args" << std::endl;
        default:
            std::cout << "Lots of args" << std::endl;
    }

    return 0;
}
```

1 arg  
2 args  
Lots of args

...Program finished with exit code 0  
Press ENTER to exit console.

# switch Statement Example

```
#include <iostream>

int main(int argc, char *argv[])
{
    switch (argc)
    {
        case 1:
            std::cout << "1 arg" << std::endl;
            break;
        case 2:
            std::cout << "2 args" << std::endl;
            break;
        default:
            std::cout << "Lots of args" << std::endl;
    }

    return 0;
}
```

1 arg

...Program finished with exit code 0  
Press ENTER to exit console.

# try/throw/catch Statements

- A **try** statement handles exceptions.

- Syntax:

```
try
{
    <statement>s;
    throw <expression>;
    ...
}
catch (<declaration>)
{
    <statement>s;
}
catch (<declaration>)
{
    <statement>s;
}
...
```

# try/throw/catch Statement

## Example 1

```
#include <iostream>

int main(int argc, char *argv[])
{
    try
    {
        if (argc == 1) throw argv[0];
        std::cout << "All done." << std::endl;
    }
    catch (const char *Msg)
    {
        std::cout << "Error Msg: " << Msg << std::endl;
    }
    catch (const int i)
    {
        std::cout << "Error Number: " << i << std::endl;
    }
    catch (...)
    {
        std::cout << "Unknown error" << std::endl;
        throw;
    }

    return 0;
}
```

Error Msg: /home/a.out

...Program finished with exit code 0  
Press ENTER to exit console.

# try/throw/catch Statement

## Example 2

```
#include <iostream>

int main(int argc, char *argv[])
{
    try
    {
        if (argc == 1) throw argc;
        std::cout << "All done." << std::endl;
    }
    catch (const char *Msg)
    {
        std::cout << "Error Msg: " << Msg << std::endl;
    }
    catch (const int i)
    {
        std::cout << "Error Number: " << i << std::endl;
    }
    catch (...)
    {
        std::cout << "Unknown error" << std::endl;
        throw;
    }

    return 0;
}
```

Error Number: 1

...Program finished with exit code 0  
Press ENTER to exit console.

# try/throw/catch Statement

## Example 2

```
#include <iostream>

int main(int argc, char *argv[])
{
    try
    {
        if (argc == 1) throw 1.0;
        std::cout << "All done." << std::endl;
    }
    catch (const char *Msg)
    {
        std::cout << "Error Msg: " << Msg << std::endl;
    }
    catch (const int i)
    {
        std::cout << "Error Number: " << i << std::endl;
    }
    catch (...)
    {
        std::cout << "Unknown error" << std::endl;
        throw;
    }

    return 0;
}
```

Unknown error  
terminate called after throwing an  
instance of 'double'  
Aborted

...Program finished with exit code 134  
Press ENTER to exit console.