Introduction to C++

• C++ Basics
• First Program
• Identifiers
• Data Types
• Variables and Constants
C++ Language History

- **1967**: "BCPL" Language (Martin Richards, Cambridge)
- **1970**: "B" Language (Ken Thompson, Bell Labs): Used to write first UNIX
- **1972**: "C" Language (Dennis Ritchie Bell Labs)
- **1985**: “C++” (Bjarne Stroustrup Bell Labs): Object oriented extension to C
  “++” connotes increment
- **1998** **ISO/ANSI Standard C++**
  - International standard enables portability
  - What we use in this course
```cpp
#include <iostream>

using namespace std;

int main ()
{
    return 0;
}
```
#include <iostream>
using namespace std;

int main()
{
    return 0;
}

where it all starts
First Program

```cpp
#include <iostream>
using namespace std;

int main ()
{
    return 0;
}
```

where it all starts

values passed to the program
#include <iostream>     //library
using namespace std;

int main()
{
    return 0;
}

values passed to the program
where it all starts

go back to where 'main' was called from
```cpp
#include <iostream>     //library
using namespace std;

int main ()
{
    return 0;
}
```

where it all starts

values passed to the program

goto back to where 'main' was called from

return value
```cpp
#include <iostream>     //library
using namespace std;

int main ()
{
    return 0;
}
```

- **return type**
- **where it all starts**
- **values passed to the program**
- **return value**
- **go back to where 'main' was called from**
```
#include <iostream>

using namespace std;

int main ()
{
    return 0;
}
```
int main ()
{
    return 0;
}

#include <iostream>
using namespace std;

int main ()
{
    return 0;
}

First Program
int main ()
{
    return 0;
}

First Program
```cpp
#include <iostream>
using namespace std;

int main()
{
    return 0;
}
```
```cpp
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    return 0;
}
```
#include <iostream>     //library
using namespace std;

int main ()
{
    cout << “Hello World!” << endl;
    return 0;
}
#include <iostream> //library
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    return 0;
}
C++ Libraries

- C++ provides a base set of standard libraries
- Libraries have pre-written code that programs can use
  - E.g., library `iostream` defines features for input & output
- Libraries need to be specifically included to be used
#include <iostream>    //library
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    return 0;
}
```cpp
#include <iostream>     //library
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    return 0;
}
```
#include <iostream>     //library
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    return 0;
}

Let’s add Output
```cpp
#include <iostream>     //library
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    return 0;
}
```
Commenting the Program

// Program to output a message
// Author: 183 Student
// Date: 09.09.2011

#include <iostream>     //library
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    return 0;
}
// Program to output a message
// Author: 183 Student
// Date: 09.09.2011

#include <iostream>  //library
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    return 0;
}
Example Program

```cpp
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    return 0;
}
```
#include <iostream>
using namespace std;

int main ()
{
    return 0;
}
Example Program

#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    return 0;
}
Always keep your code in a working state

Write some code, compile it and test it
Then write some more code, compile it and test it
Etc.

You don’t want to write a lot of code and then discover that something is wrong!

Why???
Common "operators" to do calculations are:

- Add 2 numbers
- Subtract 2 numbers
- Multiply 2 numbers
- Divide one number by another
- Remainder with INTEGER division (MOD)
Note: NO operator for exponentiation

- ^ does NOT indicate exponentiation
- ^ is Bitwise Exclusive-Or
```cpp
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    cout << 3+4*7 << endl;
    return 0;
}
```
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    cout << 3+4*7 << endl;
    return 0;
}
#include <iostream>
using namespace std;

int main ()
{
    cout << "Hello World!" << endl;
    cout << 3+4*7 << endl;
    cout << "3+4*7" << endl;
    return 0;
}
# Precedence Table

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>ASSOCIATIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGH</td>
<td>- done first</td>
</tr>
<tr>
<td>( )</td>
<td>left to right</td>
</tr>
<tr>
<td>+</td>
<td>unary; right to left</td>
</tr>
<tr>
<td>-</td>
<td>left to right</td>
</tr>
<tr>
<td>*</td>
<td>binary; left to right</td>
</tr>
<tr>
<td>/</td>
<td>right to left</td>
</tr>
<tr>
<td>%</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td></td>
</tr>
<tr>
<td>LOW</td>
<td>- done last</td>
</tr>
</tbody>
</table>

*HIGH* - done first

*LOW* - done last
# Precedence Table

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>ASSOCIATIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HIGH</strong> - done first</td>
<td></td>
</tr>
<tr>
<td>( )</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>unary; right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>binary; left to right</td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
</tr>
<tr>
<td><strong>LOW</strong>  - done last</td>
<td></td>
</tr>
<tr>
<td>5 + -(3 - 1) * 6 / 3</td>
<td></td>
</tr>
</tbody>
</table>
# Precedence Table

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>ASSOCIATIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HIGH</strong></td>
<td>- done first</td>
</tr>
<tr>
<td>( )</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>unary; right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>binary; left to right</td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
</tr>
<tr>
<td><strong>LOW</strong></td>
<td>- done last</td>
</tr>
</tbody>
</table>

5 + -(3 - 1) * 6 / 3

- **add**
- **unary**
# Precedence Table

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>ASSOCIATIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HIGH</strong></td>
<td>- done first</td>
</tr>
<tr>
<td>( )</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>unary; right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>binary; left to right</td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
</tr>
<tr>
<td><strong>LOW</strong></td>
<td>- done last</td>
</tr>
</tbody>
</table>

\[
5 + -(3 - 1) \times \frac{6}{3}
\]
## Precedence Table

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>ASSOCIATIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HIGH</strong> - done first</td>
<td></td>
</tr>
<tr>
<td>( )</td>
<td>left to right</td>
</tr>
<tr>
<td>+, -</td>
<td>unary; right to left</td>
</tr>
<tr>
<td>*, /, %</td>
<td>left to right</td>
</tr>
<tr>
<td>+, -</td>
<td>binary; left to right</td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
</tr>
<tr>
<td><strong>LOW</strong>  - done last</td>
<td></td>
</tr>
<tr>
<td>5 + -(3 - 1) * 6 / 3</td>
<td></td>
</tr>
<tr>
<td>5 – 2* 6/3</td>
<td></td>
</tr>
</tbody>
</table>
## Precedence Table

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>ASSOCIATIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HIGH</strong> - done first</td>
<td></td>
</tr>
<tr>
<td>( )</td>
<td>left to right</td>
</tr>
<tr>
<td>+</td>
<td>unary; right to left</td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>left to right</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>binary; left to right</td>
</tr>
<tr>
<td>+</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>right to left</td>
</tr>
<tr>
<td>=</td>
<td></td>
</tr>
</tbody>
</table>

| **LOW** - done last |               |
| 5 + - (3 - 1) * 6 / 3 |               |
| 5 – 2* 6/3 |               |
| 5 – 12/3 |               |
# Precedence Table

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>ASSOCIATIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HIGH</strong> - done first</td>
<td></td>
</tr>
<tr>
<td>( )</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>unary; right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>binary; left to right</td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
</tr>
<tr>
<td><strong>LOW</strong> - done last</td>
<td>5 + -(3 - 1) * 6 / 3</td>
</tr>
<tr>
<td></td>
<td>5 – 2* 6/3</td>
</tr>
<tr>
<td></td>
<td>5 – 12/3</td>
</tr>
<tr>
<td></td>
<td>5 – 4</td>
</tr>
</tbody>
</table>
# Precedence Table

<table>
<thead>
<tr>
<th>OPERATOR</th>
<th>ASSOCIATIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HIGH</strong> - done first</td>
<td></td>
</tr>
<tr>
<td>( )</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>unary; right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>binary; left to right</td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
</tr>
<tr>
<td><strong>LOW</strong>  - done last</td>
<td></td>
</tr>
<tr>
<td>5 + -(3 - 1) * 6 / 3</td>
<td></td>
</tr>
<tr>
<td>5 – 2* 6/3</td>
<td></td>
</tr>
<tr>
<td>5 – 12/3</td>
<td></td>
</tr>
<tr>
<td>5 – 4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Division can be Tricky!

- watch out if you have int / int
  - 5 / 2 is ?
  - 2 / 3 is 0
  - 5.0 / 2.0 is 2.5
  - 2.0 / 3.0 is 0.666…
  - 5 % 2 is 1
  - 2 % 3 is 2
Division can be Tricky!

- watch out if you have int / int
  - 5 / 2 is 2
  - 2 / 3 is 0
  - 5.0 / 2.0 is 2.5
  - 2.0 / 3.0 is 0.666…
  - 5 % 2 is 1
  - 2 % 3 is 2
Division can be Tricky!

- watch out if you have int / int
  - $5 / 2$ is 2
  - $2 / 3$ is $\frac{2}{3}$ or approximately 0.666...
  - $5 \% 2$ is 1
  - $2 \% 3$ is 2
Division can be Tricky!

- watch out if you have int / int
  - 5 / 2 is 2
  - 2 / 3 is 0
Division can be Tricky!

- watch out if you have int / int
  - 5 / 2 is 2
  - 2 / 3 is 0
  - 5.0 / 2.0 is ?
Division can be Tricky!

- watch out if you have int / int

  - 5 / 2 is 2
  - 2 / 3 is 0
  - 5.0 / 2.0 is 2.5
Division can be Tricky!

- watch out if you have int / int

  - $5 / 2$ is 2
  - $2 / 3$ is 0

  - $5.0 / 2.0$ is 2.5
  - $2.0 / 3.0$ is ?
Division can be tricky!

- Watch out if you have int / int

  - 5 / 2 is 2
  - 2 / 3 is 0

  - 5.0 / 2.0 is 2.5
  - 2.0 / 3.0 is 0.666...
Division can be tricky!

- watch out if you have int / int

  - 5 / 2 is 2
  - 2 / 3 is 0
  - 5.0 / 2.0 is 2.5
  - 2.0 / 3.0 is 0.666...

- 5 % 2 is ?
Division can be Tricky!

- watch out if you have int / int
  - 5 / 2 is 2
  - 2 / 3 is 0
  - 5.0 / 2.0 is 2.5
  - 2.0 / 3.0 is 0.666...
  - 5 % 2 is 1
Division can be Tricky!

- watch out if you have int / int

  - 5 / 2 is 2
  - 2 / 3 is 0
  - 5.0 / 2.0 is 2.5
  - 2.0 / 3.0 is 0.666...

  - 5 % 2 is 1
  - 2 % 3 is ?
Division can be Tricky!

- watch out if you have int / int

  - 5 / 2 is 2
  - 2 / 3 is 0

  - 5.0 / 2.0 is 2.5
  - 2.0 / 3.0 is 0.666...

  - 5 % 2 is 1
  - 2 % 3 is 2
How many hours and minutes are there in 375 minutes?
How many hours and minutes are there in 375 minutes?

375 / 60
How many hours and minutes are there in 375 minutes?

375 / 60 = 6 hours
How many hours and minutes are there in 375 minutes?

- $375 \div 60 = 6$ hours
- $375 \mod 60$
How many hours and minutes are there in 375 minutes?

- 375 / 60 = 6 hours
- 375 % 60 = 15 minutes

/ and % are useful
Suppose courseNum is a 3-digit int, with the 2\textsuperscript{nd} digit representing whether a course is required (even) or elective (odd)

Question: Find the 2\textsuperscript{nd} digit.
- E.g., if courseNum is 253 then find the ‘5’:
Suppose courseNum is a 3-digit int, with the 2nd digit representing whether a course is required (even) or elective (odd)

Question: Find the 2nd digit.
- E.g., if courseNum is 253 then find the ‘5’:

\[
\frac{253}{10} \mod 10 = 25 \mod 10 = 5
\]
Suppose courseNum is a 3-digit int, with the 2nd digit representing whether a course is required (even) or elective (odd).

Question: Find the 2nd digit.

- E.g., if courseNum is 253 then find the ‘5’:

\[
\frac{253}{10} \mod 10 = 25 \mod 10 = 5
\]

\[(\text{courseNum} / 10) \mod 10 \]
Variables

- A location in memory
- Store and retrieve a value

- Similar to algebra
  - \( x + y = 10 \)
  - \( x - y = 6 \)

- But different
Variables

- Have a **name**
  - Describes purpose

- And **type** of values it will hold
  - Domain of the values

- `int playerScore`
- `int numZombiesDefeated`
Variables

- Have a **name**
  - Describes purpose

- And **type** of values it will hold
  - Domain of the values

- `int playerScore`
- `int numZombiesDefeated`
Datatype: Type of values

- int
- char
- bool
- double

- float
  - hold over from years ago
  - computers had limited memory
Datatype: int

- Whole numbers
  - 1
  - 1000
  - -42
  - 0

- No commas (1,000 won’t work!)
- No spaces (1 000 000 won’t work!)
Datatype: char

- **char**
  - Syntax: A character is enclosed in single quotes
    - E.g., 'A'  '$'  '5'  'e'
  - Represented in computer by numeric (ASCII) codes
    - E.g., 'A' is stored as 65
  - a numeric (integral) type
    - -128 to 127
    - 1 byte
<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>Null</td>
<td>1</td>
<td>01</td>
<td>!</td>
<td>2</td>
<td>02</td>
<td>&quot;</td>
<td>3</td>
<td>03</td>
<td>#</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>$</td>
<td>5</td>
<td>05</td>
<td>%</td>
<td>6</td>
<td>06</td>
<td>$</td>
<td>7</td>
<td>07</td>
<td>'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>(</td>
<td>9</td>
<td>09</td>
<td>)</td>
<td>10</td>
<td>0A</td>
<td>*</td>
<td>11</td>
<td>0B</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>,</td>
<td>13</td>
<td>0D</td>
<td>-</td>
<td>14</td>
<td>0E</td>
<td>.</td>
<td>15</td>
<td>0F</td>
<td>/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>0</td>
<td>17</td>
<td>11</td>
<td>1</td>
<td>18</td>
<td>12</td>
<td>2</td>
<td>19</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>4</td>
<td>21</td>
<td>15</td>
<td>5</td>
<td>22</td>
<td>16</td>
<td>6</td>
<td>23</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>8</td>
<td>25</td>
<td>19</td>
<td>9</td>
<td>26</td>
<td>1A</td>
<td>:</td>
<td>27</td>
<td>1B</td>
<td>;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>&lt;</td>
<td>29</td>
<td>1D</td>
<td>=</td>
<td>30</td>
<td>1E</td>
<td>&gt;</td>
<td>31</td>
<td>1F</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Datatype: `char`

- `char`
  - Syntax: A character is enclosed in single quotes
    - E.g., 'A'  '$'  '5'  'e'
  - Represented in computer by numeric (ASCII) codes
    - E.g., 'A' is stored as 65
  - a numeric (integral) type
  - -128 to 127
  - 1 byte
Datatype: char

char
1 byte
-128 to 127

1 byte = 8 bits
Each bit is either a 1 or 0

000000001 is 1
000000010 is 2
Bits and bytes and values

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000001</td>
<td>1</td>
</tr>
<tr>
<td>000000010</td>
<td>2</td>
</tr>
<tr>
<td>000000111</td>
<td>3</td>
</tr>
<tr>
<td>000001000</td>
<td>4</td>
</tr>
<tr>
<td>000001100</td>
<td>5</td>
</tr>
<tr>
<td>000001110</td>
<td>6</td>
</tr>
<tr>
<td>000010000</td>
<td>7</td>
</tr>
<tr>
<td>000010010</td>
<td>8</td>
</tr>
<tr>
<td>000010100</td>
<td>9</td>
</tr>
<tr>
<td>000010110</td>
<td>10</td>
</tr>
<tr>
<td>000011000</td>
<td>11</td>
</tr>
<tr>
<td>000011010</td>
<td>12</td>
</tr>
<tr>
<td>000011100</td>
<td>13</td>
</tr>
<tr>
<td>000011110</td>
<td>14</td>
</tr>
<tr>
<td>000100000</td>
<td>15</td>
</tr>
<tr>
<td>000100010</td>
<td>16</td>
</tr>
</tbody>
</table>
# Bits and bytes and values

<table>
<thead>
<tr>
<th>Bin</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000001</td>
<td>1</td>
</tr>
<tr>
<td>00000000010</td>
<td>2</td>
</tr>
<tr>
<td>00000000110</td>
<td>3</td>
</tr>
<tr>
<td>00000010000</td>
<td>4</td>
</tr>
<tr>
<td>00001010010</td>
<td>5</td>
</tr>
<tr>
<td>00001010101</td>
<td>6</td>
</tr>
<tr>
<td>00001100000</td>
<td>7</td>
</tr>
<tr>
<td>00001100001</td>
<td>8</td>
</tr>
<tr>
<td>00001100010</td>
<td>9</td>
</tr>
<tr>
<td>00001100011</td>
<td>10</td>
</tr>
<tr>
<td>00001100100</td>
<td>11</td>
</tr>
<tr>
<td>00001100101</td>
<td>12</td>
</tr>
<tr>
<td>00001101000</td>
<td>13</td>
</tr>
<tr>
<td>00001101001</td>
<td>14</td>
</tr>
<tr>
<td>00001101010</td>
<td>15</td>
</tr>
<tr>
<td>00001101011</td>
<td>16</td>
</tr>
</tbody>
</table>
# Bits and bytes and values

<table>
<thead>
<tr>
<th>Binary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000001</td>
<td>1</td>
</tr>
<tr>
<td>0000000010</td>
<td>2</td>
</tr>
<tr>
<td>0000000110</td>
<td>3</td>
</tr>
<tr>
<td>0000001000</td>
<td>4</td>
</tr>
<tr>
<td>0000010110</td>
<td>5</td>
</tr>
<tr>
<td>0000110110</td>
<td>6</td>
</tr>
<tr>
<td>0001111010</td>
<td>7</td>
</tr>
<tr>
<td>0001111110</td>
<td>8</td>
</tr>
<tr>
<td>0010000011</td>
<td>9</td>
</tr>
<tr>
<td>0010000100</td>
<td>10</td>
</tr>
<tr>
<td>0010001000</td>
<td>11</td>
</tr>
<tr>
<td>0010010110</td>
<td>12</td>
</tr>
<tr>
<td>0010101010</td>
<td>13</td>
</tr>
<tr>
<td>0010110010</td>
<td>14</td>
</tr>
<tr>
<td>0011011110</td>
<td>15</td>
</tr>
<tr>
<td>0011101110</td>
<td>16</td>
</tr>
</tbody>
</table>
## Bits and bytes and values

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000001</td>
<td>1</td>
</tr>
<tr>
<td>00000000010</td>
<td>2</td>
</tr>
<tr>
<td>00000000110</td>
<td>3</td>
</tr>
<tr>
<td>00000010000</td>
<td>4</td>
</tr>
<tr>
<td>00000101010</td>
<td>5</td>
</tr>
<tr>
<td>00001110000</td>
<td>6</td>
</tr>
<tr>
<td>00111011000</td>
<td>7</td>
</tr>
</tbody>
</table>

16 bit values
# Bits and bytes and values

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0</td>
<td>2</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 1</td>
<td>3</td>
</tr>
<tr>
<td>0 0 0 0 0 1 0 0</td>
<td>4</td>
</tr>
<tr>
<td>0 0 0 0 0 1 0 1</td>
<td>5</td>
</tr>
<tr>
<td>0 0 0 0 0 1 1 0</td>
<td>6</td>
</tr>
<tr>
<td>0 0 0 0 1 1 1</td>
<td>7</td>
</tr>
<tr>
<td>0 0 0 1 0 0 0</td>
<td>8</td>
</tr>
</tbody>
</table>
# Bits and bytes and values

<table>
<thead>
<tr>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0</td>
<td>2</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 1</td>
<td>3</td>
</tr>
<tr>
<td>0 0 0 0 0 1 0 0</td>
<td>4</td>
</tr>
<tr>
<td>0 0 0 0 0 1 0 1</td>
<td>5</td>
</tr>
<tr>
<td>0 0 0 0 1 1 0</td>
<td>6</td>
</tr>
<tr>
<td>0 0 0 0 1 1 1</td>
<td>7</td>
</tr>
<tr>
<td>0 0 0 1 0 0 0</td>
<td>8</td>
</tr>
<tr>
<td>0 0 0 1 0 0 1</td>
<td>9</td>
</tr>
<tr>
<td>0 0 0 1 0 1 0</td>
<td>10</td>
</tr>
<tr>
<td>0 0 0 1 0 1 1</td>
<td>11</td>
</tr>
<tr>
<td>0 0 0 1 1 0 0</td>
<td>12</td>
</tr>
<tr>
<td>0 0 0 1 1 1 0</td>
<td>13</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0</td>
<td>14</td>
</tr>
<tr>
<td>0 0 1 0 0 0 1</td>
<td>15</td>
</tr>
<tr>
<td>0 1 0 0 0 0 0</td>
<td>16</td>
</tr>
</tbody>
</table>
### Bits and bytes and values

<table>
<thead>
<tr>
<th>Binary</th>
<th>Value</th>
<th>Binary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000000001</td>
<td>1</td>
<td>0000010010</td>
<td>9</td>
</tr>
<tr>
<td>0000000100</td>
<td>2</td>
<td>0000010100</td>
<td>10</td>
</tr>
<tr>
<td>0000001111</td>
<td>3</td>
<td>0000010110</td>
<td>11</td>
</tr>
<tr>
<td>0000011000</td>
<td>4</td>
<td>0000011000</td>
<td>12</td>
</tr>
<tr>
<td>0000011011</td>
<td>5</td>
<td>0000011011</td>
<td>13</td>
</tr>
<tr>
<td>0000011100</td>
<td>6</td>
<td>0000011100</td>
<td>14</td>
</tr>
<tr>
<td>0000011111</td>
<td>7</td>
<td>0000011111</td>
<td>15</td>
</tr>
<tr>
<td>0000100000</td>
<td>8</td>
<td>0001000000</td>
<td>16</td>
</tr>
</tbody>
</table>
Datatype: char

- **char**
  - Syntax: A character is enclosed in single quotes
    - E.g., 'A'  '$'  '5'  'e'
  - Represented in computer by numeric (ASCII) codes
    - E.g., ‘A’ is stored as 65
  - a numeric (integral) type
    - -128 to 127
    - 1 byte
Datatype: **int**

**int**

Usually 4 bytes

-2147483468 to 2147483467

Size depends on system
#include <climits>

using namespace std;

int main (void)
{
    cout << INT_MIN << endl;
    cout << INT_MAX << endl;
    return 0;
}
#include <climits>

int main (void)
{
    cout << INT_MIN << endl;
    cout << INT_MAX << endl;
    cout << CHAR_MIN << endl;
    cout << CHAR_MAX << endl;
    return 0;
}
**Datatype: double**

**double**

Usually 8 bytes

±2.22507e-308 to 1.79769e+308

"default" datatype for Real Numbers

Size depends on system
Some examples of double

1.0
1.
3.141592
-42.15
1.3e12 \quad (1.3 \times 10^{12})
3.2E4 \quad (3.2 \times 10^4)
Datatype: float

**Float**

Usually 4 bytes

$\pm 1.17549 \times 10^{-38}$ to $3.40282 \times 10^{38}$
float/double sizes/ranges

#include <cfloat>

cout << FLT_MIN << endl;
cout << FLT_MAX << endl;
cout << DBL_MIN << endl;
cout << DBL_MAX << endl;
Datatype: bool

- bool
  - true or false
Question

- Name 4 datatypes
Name 4 datatypes

- int
- double
- char
- bool
- float
Variables: Remember

- Variables must have a name
- Variables must have a type
Variables must have a name
Variables must have a type

Examples:
- `int value;`
- `double pi;`
- `int a, b, c;`
- `char oneChar;`
Variable / Identifier Rules

1) Start with a letter or underscore ('_')
2) After the first character, any number of letters, underscores, or digits
3) Can’t be reserved word

- Max size 31 chars (most systems)
- C++ is case-sensitive (int != Int)
- System identifiers usually start with '_'
What does a Declaration do?

- Sets up a storage location
- Does not put any value there yet

```c
int value;
```
What does a Declaration do?

- Sets up a storage location
- Does not store any value there yet

```c
int value;
double pi;
```
int sum;
sum = 0;

double pi = 3.14;

char val = 'G';
Assign what is on the right into variable on the left

```plaintext
value = 4;
x = 3;
y = x * x;
y = y / value;
```
Assign what is on the right into variable on the left

value ← 4;
x ← 3;
y ← x * x;
y ← y / value;

- Done in order
- Not formulas
- Calculate RHS
- Assign result into LHS
  - (sometimes referred to as lvalue)
Assignment Statements

Assign what is on the right into variable on the left

```plaintext
value = 4;
x = 3;
y = x * x;
y = y / value;
```

- Done in order
- Not formulas
- Calculate RHS
- Assign result into LHS
  - (sometimes referred to as lvalue)
value = 4;
x = 3;
y = x * x;
y = y / value;
value = 4;
x = 3;
y = x * x;
y = y / value;
value = 4;
x = 3;
y = x * x;
y = y / value;
value = 4;
x = 3;
y = x * x;
y = y / value;
Assignment Statements

```plaintext
value = 4;
x = 3;
y = x * x;
y = y / value;
```
value = 4;
x = 3;
y = x * x;
y = y / value;
value = 4;
x = 3;
y = x * x;
y = y / value;
value = 4;
x = 3;
y = x * x;
y = y / value;
What prints?

```c++
int x;
cout << x << endl;
```

Prints garbage?

That is, prints whatever arbitrary value happens to be in x.
int x;
cout << x << endl;

Prints garbage
That is, prints whatever arbitrary value happens to be in x
What happens?

```cpp
int giraffe = 23;
int rabbit = 3;

rabbit = giraffe;

cout << rabbit << endl;
```
double giraffe = 23456.789;
int rabbit = 3;

rabbit = giraffe;

cout << rabbit << endl;
You really don’t want to spill that…

You can’t put a large value into a small cup

- Well, you can, but you’ll lose some
- Called "spillage"
- Compiler tries to help prevent this
  - Issues warning
  - Pay attention to all "warnings"
#include <iostream>
using namespace std;

int main()
{
    double x = 6.4;
    int a = x;
    cout << a << endl;
    return 0;
}
```cpp
#include <iostream>
using namespace std;

int main()
{
    double x = 6.4;
    int a = x;
    cout << a << endl; // prints 6
    return 0; // the decimal portion spills
}
```
Larger will NOT fit into smaller
Something will spill and will be lost

sizeof:

bool = char ≤ int ≤ float ≤ double
Smaller fits into larger
With room to spare

```cpp
int x = 5;
double y;

y = x;
cout << y << endl;
```
What is stored?

```java
double someDouble;
someDouble = 12;
```
What is stored?

double someDouble;

double someDouble = 12;

someDouble
What is stored?

double someDouble;

someDouble = 12;

Causes implicit type conversion
What is stored?

```java
int someInt;

someInt = 4.8;
```
What is stored?

```plaintext
int someInt;

someInt = 4.8;
```

Causes implicit type conversion
What is stored?

```c
int someInt;

someInt = 4.8;
```

Causes implicit type conversion
Overflow

1...2...

BAAA

1,306...1,307...

BAAA

32,767...-32,768...

BAAA BAAA BAAA

-32,767...-32,766...

BAAA
Check it out
Type Conversion in Assignment

double dbl;
int i;
char ch;

i = 25;
ch = '$';
dbl = 0.5;
dbl = 4;
i = 3.7;
double dbl;
int i;
char ch;

i = 25;    // int ← int
ch = '$';  // char ← char
dbl = 0.5; // double ← double
dbl = 4;   // double ← int
i = 3.7;   // int ← double
    // truncated
In a mixed type expression,
- Each sub-expression is *promoted* to the "highest" type prior to evaluation

Type Promotion Guidelines:
- bool = char ≤ int ≤ double

What is the type of:
- `'A' + 3 * 5.71` ?
More Example Assignments

double x;

x = 11 / 5;

x = 11.0 / 5;
double x;

x = 11 / 5; // x ← 2.0

x = 11.0 / 5;
double x;

x = 11 / 5;    // x ← 2.0

x = 11.0 / 5;
double x;

x = 11 / 5;       // x ← 2.0

x = 11.0 / 5;     // x ← 2.2
Common Assignment Errors

double x(5.2), y(7.5);
char ch = 'A';
int i = 10, j = 42;

\texttt{x = x \% y;}

ch = $;

i = 5;
j = 0;
x = i / j;
double x(5.2), y(7.5);
char ch = 'A';
int i = 10, j = 42;

x = x % y;       // syntax error

ch = $;

i = 5;
j = 0;
x = i / j;
double x(5.2), y(7.5);
char ch = 'A';
int i = 10, j = 42;

x = x % y;       // syntax error

ch = $;

i = 5;
j = 0;
x = i / j;
Common Assignment Errors

double x(5.2), y(7.5);
char ch = 'A';
int i = 10, j = 42;

x = x % y;        // syntax error
ch = $;           // syntax error

i = 5;
j = 0;
x = i / j;
double x(5.2), y(7.5);
char ch = 'A';
int i = 10, j = 42;

x = x % y; // syntax error

ch = $; // syntax error

i = 5;
j = 0;
x = i / j;
Common Assignment Errors

double x(5.2), y(7.5);
char ch = 'A';
int i = 10, j = 42;

x = x % y; // syntax error
ch = $; // syntax error
i = 5;
j = 0;
x = i / j; // run-time error
Type Casting

- We can explicitly convert types using casting operators

- Syntax -- old form:
  
  (int) x
  (double) i * 3

- Syntax -- new form:
  
  static_cast<int> (x)
  static_cast<double>(i * 3)
Cast examples: old form

```c
int i = 1, j = 3;

double x = i / j;
```
Cast examples: old form

```c
int i = 1, j = 3;
double x = i / j;  // x ← 0.0
```
int i = 1, j = 3;

double x = i / j;  // x ← 0.0

x = (double) i / (double) j;
Cast examples: old form

```c
int i = 1, j = 3;

double x = i / j;    // x ← 0.0

x = (double) i / (double) j; // x ← 0.33
```
int i = 1, j = 3;

double x = i / j;    // x ← 0.0

x = (double) i / (double) j; // x ← 0.33

x = (double) (i / j);
Cast examples: old form

```c
int i = 1, j = 3;

double x = i / j; // x ← 0.0

x = (double) i / (double) j; // x ← 0.33

x = (double) (i / j); // x ← 0.0
```
Cast examples: new form

int i = 1, j = 3;

double x = i / j; // x ← 0.0

x = static_cast<double>(i) /
static_cast<double>(j); // x ← 0.33

x = static_cast<double>(i / j); // x ← 0.0
Fun with Casting

double x = 3.7;

i = (int) (x);
double x = 3.7;

i = (int) (x); // i ← 3
Fun with Casting

double x = 3.7;

i = (int) (x);           // i ← 3

i = (int) (x + 0.5);   

Fun with Casting

double x = 3.7;

i = (int) (x); // i ← 3

i = (int) (x + 0.5); // i ← 4
Fun with Casting

double x = 3.7;

i = (int) (x); // i ← 3

i = (int) (x + 0.5); // i ← 4 (rounds x)
Outputting a variable’s value

- To output a variable’s value, you just need to put it in the `cout` statement.
- "" must not be used.

```cpp
int a = 4;
cout << a << endl;
```
Outputting a variable’s value

- To output a variable’s value, you just need to put it in the `cout` statement

- “ ” must not be used

```cpp
int a = 4;
cout << a << endl;
cout << "a" << endl;
```
From the following list, circle the statements that would be legal if these lines were in a single main program:

```c
int x = 34.5;
bool boo = x;
int g = 17;
int y = g;
y = y + 10;
int s;
s = y;
int b = 3;
int v = b;
int n = 12;
v = n;
double k = 1000 * 1000 * 1000 * 90 * .5;
double y = 9.5;
int p = 3 * g + y;
```
Across
3. Can't pin it down
4. Acronym for a chip
7. What's a prompt good for?
8. Just gotta have one
10. RUN
13. You're never going to change!!!
14. Could be called "Father"

Down
1. Quite a crew of characters
2. Not an integer (or ____ your boat)
3. Nothing is there
5. Source code consumer
6. Acronym for your laptop's power
9. Announce a new variable
11. Number variable type
12. Department of LAN jockeys
13. Say something