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
int main (void)
{
    return 0;
}
where it all starts

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

using namespace std;

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

**First Program**

where it all starts

values passed to the program
First Program

```cpp
#include <iostream>

using namespace std;

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

where it all starts

values passed to the program

go back to where 'main' was called from
```c++
#include <iostream>     // library

using namespace std;

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

where it all starts

values passed to the program

go back to where 'main' was called from

return value
```cpp
#include <iostream>

using namespace std;

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

**First Program**

- **Return type**: The function type is `int`.
- **Where it all starts**: The program execution starts with the `main` function.
- **Values passed to the program**: The program receives values through the arguments passed to the `main` function.
- **Return value**: The program returns an integer value.
- **Go back to where 'main' was called from**: After returning the value, the program control is transferred back to the caller.
```c++
#include <iostream>

using namespace std;

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

- **return type**: int
- **where it all starts**: int main (void)
- **values passed to the program**: values passed
- **return value**: return 0
- **go back to where 'main' was called from**: return value

**Note**: These MUST match in types.
#include <iostream>

using namespace std;

int main (void)
{
    return 0;
}

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

int main (void)
{
    return 0;
}

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

int main (void)
{
    return 0;
}

First Program
```cpp
#include <iostream>

using namespace std;

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

int main (void)
{
    cout << “Hello World!” << endl;
    return 0;
}
Let’s add Output

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

int main (void)
{
    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
Let’s add Output

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

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

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

int main (void)
{
    cout << "Hello World!" << endl;
    return 0;
}
Let’s add Output

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

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

- 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
#include <iostream>
using namespace std;

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

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

int main (void)
{
    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><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>
## 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>=</td>
<td></td>
</tr>
<tr>
<td><strong>LOW</strong>  - done last</td>
<td>5 + -(3 - 1) * 6 / 3</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></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></td>
</tr>
<tr>
<td>-</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) * 6 / 3
### 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>
<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></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>

\[
5 + -(3 - 1) * 6 / 3 \\
5 - 2* 6/3 \\
5 - 12/3
\]
# 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></td>
</tr>
</tbody>
</table>

\[ 5 + -(3 - 1) \times 6 \div 3 \]
\[ 5 - 2 \times 6/3 \]
\[ 5 - 12/3 \]
\[ 5 - 4 \]
# Precedence Table

**OPERATOR** | **ASSOCIATIVITY**
--- | ---
**HIGH**  | `- done first`
( ) | left to right
+ - | unary; right to left
* / % | left to right
+ - | binary; left to right
= | right to left

**LOW**  | `- done last`
5 + -(3 - 1) * 6 / 3
5 – 2* 6/3
5 – 12/3
5 – 4
1
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 \( 0.666… \)
  - \( 5 \mod 2 \) is \( 1 \)
  - \( 2 \mod 3 \) is \( 2 \)
Division can be Tricky!

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

- watch out if you have int / int
  - $5 / 2$ is $2$
  - $2 / 3$ is $?$
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
  - \(5 \div 2\) is 2
  - \(2 \div 3\) is 0
  - \(5.0 \div 2.0\) 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
  - $5 \div 2$ is 2
  - $2 \div 3$ is 0
  - $5.0 \div 2.0$ is 2.5
  - $2.0 \div 3.0$ is 0.666...
  - $5 \% 2$ is 1
Division can be Trick!y!

- 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 ?
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 / 60

6 hours

375 % 60
/ and % are useful

- How many hours and minutes are there in 375 minutes?
  - \( \frac{375}{60} \) 6 hours
  - \( 375 \mod 60 \) 15 minutes
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’:
Or how about --- elective or required?

- 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’:

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

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

\[
\text{courseNum} \div 10 \mod 10
\]
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>32</td>
<td>20</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>8</td>
<td>96</td>
<td>60</td>
<td>\</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>Start of heading</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>61</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>Start of text</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>62</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>End of text</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>63</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>End of transmit</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>64</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>Enquiry</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>Acknowledge</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>66</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>Audible bell</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>Backspace</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>Horizontal tab</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>69</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>Line feed</td>
<td>42</td>
<td>2A</td>
<td>*</td>
<td>74</td>
<td>4A</td>
<td>J</td>
<td>106</td>
<td>6A</td>
<td>j</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>Vertical tab</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>6B</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>Form feed</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>6C</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>Carriage return</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>6D</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>Shift out</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6E</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>Shift in</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>Data link escape</td>
<td>48</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>P</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>Device control 1</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>Q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>Device control 2</td>
<td>50</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>R</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>Device control 3</td>
<td>51</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>S</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>Device control 4</td>
<td>52</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>T</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>Neg. acknowledge</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>U</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>Synchronous idle</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>V</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>End trans. block</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>W</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>Cancel</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>X</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>End of medium</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>Y</td>
<td>121</td>
<td>79</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>1A</td>
<td>Substitution</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>Z</td>
<td>122</td>
<td>7A</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>1B</td>
<td>Escape</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>[</td>
<td>123</td>
<td>7B</td>
<td>(</td>
</tr>
<tr>
<td>28</td>
<td>1C</td>
<td>File separator</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>\</td>
<td>124</td>
<td>7C</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1D</td>
<td>Group separator</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>]</td>
<td>125</td>
<td>7D</td>
<td>)</td>
</tr>
<tr>
<td>30</td>
<td>1E</td>
<td>Record separator</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>^</td>
<td>126</td>
<td>7E</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>1F</td>
<td>Unit separator</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td></td>
<td>127</td>
<td>7F</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

0 0 0 0 0 0 0 1 is 1
0 0 0 0 0 0 0 1 0 is 2
### Bits and bytes and values

<table>
<thead>
<tr>
<th>Binary</th>
<th>Value</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000100</td>
<td>5</td>
</tr>
<tr>
<td>00000110</td>
<td>6</td>
</tr>
<tr>
<td>00001000</td>
<td>7</td>
</tr>
<tr>
<td>00001100</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001101</td>
<td>9</td>
</tr>
<tr>
<td>00010001</td>
<td>10</td>
</tr>
<tr>
<td>00010011</td>
<td>11</td>
</tr>
<tr>
<td>00010101</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Binary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010110</td>
<td>13</td>
</tr>
<tr>
<td>00011000</td>
<td>14</td>
</tr>
<tr>
<td>00011010</td>
<td>15</td>
</tr>
<tr>
<td>00011100</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>0000000001</td>
<td>1</td>
</tr>
<tr>
<td>0000000010</td>
<td>2</td>
</tr>
<tr>
<td>0000000111</td>
<td>3</td>
</tr>
<tr>
<td>0000001000</td>
<td>4</td>
</tr>
<tr>
<td>0000010111</td>
<td>5</td>
</tr>
<tr>
<td>0000101001</td>
<td>6</td>
</tr>
<tr>
<td>0001001001</td>
<td>7</td>
</tr>
<tr>
<td>0010001001</td>
<td>8</td>
</tr>
<tr>
<td>0100001001</td>
<td>9</td>
</tr>
<tr>
<td>1000001001</td>
<td>10</td>
</tr>
<tr>
<td>1100001001</td>
<td>11</td>
</tr>
<tr>
<td>1110001001</td>
<td>12</td>
</tr>
<tr>
<td>1111001001</td>
<td>13</td>
</tr>
<tr>
<td>1000101001</td>
<td>14</td>
</tr>
<tr>
<td>1010101001</td>
<td>15</td>
</tr>
<tr>
<td>1101101001</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>0 0 0 0 0 0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1 0</td>
<td>2</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1 1</td>
<td>3</td>
</tr>
<tr>
<td>0 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 1 1 0</td>
<td>7</td>
</tr>
<tr>
<td>0 0 1</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
# Bits and bytes and values

<table>
<thead>
<tr>
<th>Binary</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000001</td>
<td>1</td>
</tr>
<tr>
<td>000000010</td>
<td>2</td>
</tr>
<tr>
<td>000000110</td>
<td>3</td>
</tr>
<tr>
<td>000010000</td>
<td>4</td>
</tr>
<tr>
<td>000101000</td>
<td>5</td>
</tr>
<tr>
<td>001011000</td>
<td>6</td>
</tr>
<tr>
<td>010111000</td>
<td>7</td>
</tr>
<tr>
<td>101111000</td>
<td>8</td>
</tr>
<tr>
<td>110000000</td>
<td>9</td>
</tr>
<tr>
<td>110000001</td>
<td>10</td>
</tr>
<tr>
<td>110000010</td>
<td>11</td>
</tr>
<tr>
<td>110000100</td>
<td>12</td>
</tr>
<tr>
<td>110001010</td>
<td>13</td>
</tr>
<tr>
<td>110011100</td>
<td>14</td>
</tr>
<tr>
<td>111000000</td>
<td>15</td>
</tr>
<tr>
<td>111000001</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>0 0 0 0 0 0 0 0 1</td>
<td>1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1 0</td>
<td>2</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1 1</td>
<td>3</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0</td>
<td>4</td>
</tr>
<tr>
<td>0 0 0 0 0 1 0 1 1</td>
<td>5</td>
</tr>
<tr>
<td>0 0 0 0 1 1 0 1 0</td>
<td>6</td>
</tr>
<tr>
<td>0 0 0 1 1 1 0 0 0</td>
<td>7</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0 1 1</td>
<td>8</td>
</tr>
<tr>
<td>0 1 0 0 0 0 1 1 1</td>
<td>9</td>
</tr>
<tr>
<td>1 0 0 0 0 0 1 1 0</td>
<td>10</td>
</tr>
<tr>
<td>1 0 0 1 1 0 0 0 0</td>
<td>11</td>
</tr>
<tr>
<td>1 1 0 0 0 1 1 0 1</td>
<td>12</td>
</tr>
<tr>
<td>1 1 0 1 0 0 1 1 1</td>
<td>13</td>
</tr>
<tr>
<td>1 1 1 0 0 1 1 0 0</td>
<td>14</td>
</tr>
<tr>
<td>1 1 1 1 0 1 0 0 0</td>
<td>15</td>
</tr>
<tr>
<td>1 1 1 1 1 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>
</tr>
</thead>
<tbody>
<tr>
<td>0000000001</td>
<td>1</td>
</tr>
<tr>
<td>0000000100</td>
<td>2</td>
</tr>
<tr>
<td>0000000101</td>
<td>3</td>
</tr>
<tr>
<td>0000000100</td>
<td>4</td>
</tr>
<tr>
<td>0000000111</td>
<td>5</td>
</tr>
<tr>
<td>0000000110</td>
<td>6</td>
</tr>
<tr>
<td>0000000110</td>
<td>7</td>
</tr>
<tr>
<td>0000000111</td>
<td>8</td>
</tr>
<tr>
<td>0000010000</td>
<td>9</td>
</tr>
<tr>
<td>0000010001</td>
<td>10</td>
</tr>
<tr>
<td>0000010000</td>
<td>11</td>
</tr>
<tr>
<td>0000010000</td>
<td>12</td>
</tr>
<tr>
<td>0000010000</td>
<td>13</td>
</tr>
<tr>
<td>0000010000</td>
<td>14</td>
</tr>
<tr>
<td>0000010000</td>
<td>15</td>
</tr>
<tr>
<td>0000010000</td>
<td>16</td>
</tr>
</tbody>
</table>
### Bits and bytes and values

<table>
<thead>
<tr>
<th>Bits</th>
<th>Value</th>
<th>Bits</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 1</td>
<td>1</td>
<td>0 0 0 0 0 1 0 0 1</td>
<td>9</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 0 0</td>
<td>2</td>
<td>0 0 0 0 1 0 1 0 0</td>
<td>10</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 1 0</td>
<td>3</td>
<td>0 0 0 0 1 0 1 1 0</td>
<td>11</td>
</tr>
<tr>
<td>0 0 0 0 0 1 0 0 0</td>
<td>4</td>
<td>0 0 0 0 1 1 0 0 0</td>
<td>12</td>
</tr>
<tr>
<td>0 0 0 0 1 0 1 1 0</td>
<td>5</td>
<td>0 0 0 0 1 1 0 0 1</td>
<td>13</td>
</tr>
<tr>
<td>0 0 0 1 1 0 1 1 0</td>
<td>6</td>
<td>0 0 0 1 1 1 0 0 0</td>
<td>14</td>
</tr>
<tr>
<td>0 0 1 1 1 1 1 1 1</td>
<td>7</td>
<td>0 0 1 1 1 1 1 1 1</td>
<td>15</td>
</tr>
<tr>
<td>0 0 1 0 0 0 0 0 0</td>
<td>8</td>
<td>0 0 1 0 0 0 0 0 0</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

-2,147,483,468 to 2,147,483,467

Size depends on system
#include <climits>

```cpp
cout << INT_MIN << endl;
cout << INT_MAX << endl;
```

int sizes/ranges
#include <climits>

#include <climits>

cout << INT_MIN << endl;
cout << INT_MAX << endl;
cout << CHAR_MIN << endl;
cout << CHAR_MAX << endl;
Datatype: double

- **double**
- Usually 8 bytes
- $\pm 2.22507e-308$ 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  (1.3 \times 10^{12})
3.2E4   (3.2 \times 10^{4})
Datatype: float

float

Usually 4 bytes

±1.17549e-38 to 3.40282e+38
float/double sizes/ranges

```cpp
#include <cfloat>

int main (void)
{
    cout << FLT_MIN << endl;
    cout << FLT_MAX << endl;
    cout << DBL_MIN << endl;
    cout << DBL_MAX << endl;
    return 0;
}
```
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

- 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;
```
Storing values in variables

```c
int sum;
sum = 0;

double pi = 3.14;

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

```python
value = 4;
x = 3;
y = x * x;
y = y / value;
```
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)
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)
Assignment Statements

```
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;
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;
$$\begin{align*}
\text{value} & = 4; \\
x & = 3; \\
y & = x \times x; \\
y & = y / \text{value};
\end{align*}$$
What prints?

```cpp
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
int giraffe = 23;
int rabbit = 3;
rabbit = giraffe;
cout << rabbit << endl;
What happens?

double giraffe = 23456.789;
int rabbit = 3;

rabbit = giraffe;

cout << rabbit << endl;
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( void )
{
    double x = 6.4;
    int a = x;
    cout << a << endl;
    cout << a << endl;
    return 0;
}
What prints?

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

int main( void )
{
    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;
```
double someDouble;

someDouble = 12;
What is stored?

double someDouble;

someDouble = 12;
What is stored?

double someDouble;

double someDouble = 12;

Causes implicit type conversion
What is stored?

```java
int someInt;
someInt = 4.8;
```

Causes implicit type conversion
What is stored?

```java
int someInt;

someInt = 4.8;
```

Causes implicit type conversion
What is stored?

```java
int someInt;

someInt = 4.8;
```

4

somenInt

Causes implicit type conversion
Overflow
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
double x;

x = 11 / 5;

x = 11.0 / 5;
More Example Assignments

double x;

x = 11 / 5;       // x ← 2.0

x = 11.0 / 5;
double x;

x = 11 / 5;  // x \leftarrow 2.0

x = 11.0 / 5;
double x;

x = 11 / 5;    // x ← 2.0

x = 11.0 / 5;  // x ← 2.2
double x(5.2), y(7.5);
char ch = 'A';
int i = 10, j = 42;

x = x % y;

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 = $;

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 = $; // 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
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
```
Cast examples: old form

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

double x = i / j; // x \leftarrow 0.0

x = (double) i / (double) j; // x \leftarrow 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

```cpp
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);
Fun with Casting

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

```
int a = 4;
cout << a << endl;
```

“ ” must not be used.
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;
```
Sharpen your pencil

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