<table>
<thead>
<tr>
<th>Time</th>
<th>Title</th>
<th>Presenter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day 1</td>
<td>Programming &amp; The C Language</td>
<td></td>
</tr>
<tr>
<td>Day 2</td>
<td>Parallel Programming: MPI and OpenMP</td>
<td></td>
</tr>
<tr>
<td>Day 3</td>
<td>Abstraction, Struct &amp; C++</td>
<td></td>
</tr>
<tr>
<td>9:00-9:10</td>
<td>Welcome</td>
<td>Frank</td>
</tr>
<tr>
<td>9:10-9:30</td>
<td>Git – Updating your repos</td>
<td></td>
</tr>
<tr>
<td>9:30-10:00</td>
<td>PI</td>
<td>Frank</td>
</tr>
<tr>
<td>10:00-10:30</td>
<td>Abstraction</td>
<td>Frank</td>
</tr>
<tr>
<td>10:45-11:30</td>
<td>Abstraction in C with struct and pointers</td>
<td>You</td>
</tr>
<tr>
<td>11:30-12:30</td>
<td>Exercise</td>
<td></td>
</tr>
<tr>
<td>12:30-1:30</td>
<td>LUNCH</td>
<td></td>
</tr>
<tr>
<td>1:30-2:00</td>
<td>Programming in C++</td>
<td>Frank</td>
</tr>
<tr>
<td>2:00-5:00</td>
<td>Exercise With Frank</td>
<td>Me and You</td>
</tr>
<tr>
<td>Day 4</td>
<td>User Interface Design &amp; Qt</td>
<td></td>
</tr>
<tr>
<td>Day 5</td>
<td>SimCenter &amp; Cloud Computing</td>
<td></td>
</tr>
</tbody>
</table>
Abstraction &
C Structures, the C++ Language
Frank McKenna
Outline

• Abstraction
• C Programming Language –
  • Structures
  • Container Classes

• Object-Oriented Programming
• C++
Definition – Digital Computer

“Digital computer, any of a class of devices capable of solving problems by processing information in discrete form. It operates on data, including magnitudes, letters, and symbols, that are expressed in binary code — i.e., using only the two digits 0 and 1. By counting, comparing, and manipulating these digits or their combinations according to a set of instructions held in its memory, a digital computer can perform such tasks as to control industrial processes and regulate the operations of machines; analyze and organize vast amounts of business data; and simulate the behaviour of dynamic systems (e.g., global weather patterns and chemical reactions) in scientific research.” (source: enclyopedia Britannica)
Abstraction

• “The process of removing physical, spatial, or temporal details\(^\text{[2]}\) or attributes in the study of objects or systems in order to more closely attend to other details of interest” [source: wikippedia]
We Work in Decimal

0,1,2,3,4,5,6,7,8,9
Computers in Binary

0,1

Computer Bit (on/off) (0,1)
We Combine Numbers

\[
\begin{array}{ccc}
100 & 10 & 1 \\
\end{array}
\]

\[4 \times 100 + 5 \times 10 + 6\]

With 3 numbers we can represent any number 0 through 999
What can we represent on a computer with 3 bits

$2^3$ possibilities
We Combine Numbers

With 3 numbers we can represent any number 0 through 7
What can we represent on a computer with 3 bits

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td>001</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>010</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>011</td>
<td>3</td>
<td>D</td>
</tr>
<tr>
<td>100</td>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>101</td>
<td>5</td>
<td>F</td>
</tr>
<tr>
<td>110</td>
<td>6</td>
<td>G</td>
</tr>
<tr>
<td>111</td>
<td>7</td>
<td>H</td>
</tr>
</tbody>
</table>
Computer groups bits into Bytes

1 Byte = 8 bits

$2^8 = 256$ possibilities
C Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Size</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1 byte</td>
<td>-128 to 127 or 0 to 255</td>
</tr>
<tr>
<td>unsigned char</td>
<td>1 byte</td>
<td>0 to 255</td>
</tr>
<tr>
<td>signed char</td>
<td>1 byte</td>
<td>-128 to 127</td>
</tr>
<tr>
<td>int</td>
<td>2 or 4 bytes</td>
<td>-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>unsigned int</td>
<td>2 or 4 bytes</td>
<td>0 to 65,535 or 0 to 4,294,967,295</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>unsigned short</td>
<td>2 bytes</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td>long</td>
<td>4 bytes</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4 bytes</td>
<td>0 to 4,294,967,295</td>
</tr>
</tbody>
</table>
## C Character Set

<table>
<thead>
<tr>
<th>ASCII Value</th>
<th>Character</th>
<th>Meaning</th>
<th>ASCII Value</th>
<th>Character</th>
<th>ASCII Value</th>
<th>Character</th>
<th>ASCII Value</th>
<th>Character</th>
<th>ASCII Value</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NULL</td>
<td>null</td>
<td>32</td>
<td>Space</td>
<td>64</td>
<td>@</td>
<td>96</td>
<td>~</td>
<td>97</td>
<td>a</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>Start of header</td>
<td>33</td>
<td>!</td>
<td>65</td>
<td>A</td>
<td>98</td>
<td>b</td>
<td>99</td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>start of text</td>
<td>34</td>
<td>&quot;</td>
<td>66</td>
<td>B</td>
<td>100</td>
<td>d</td>
<td>101</td>
<td>e</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>end of text</td>
<td>35</td>
<td>#</td>
<td>67</td>
<td>C</td>
<td>102</td>
<td>f</td>
<td>103</td>
<td>g</td>
</tr>
<tr>
<td>4</td>
<td>FOT</td>
<td>end of transaction</td>
<td>36</td>
<td>$</td>
<td>68</td>
<td>D</td>
<td>104</td>
<td>h</td>
<td>105</td>
<td>i</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>enquiry</td>
<td>37</td>
<td>%</td>
<td>69</td>
<td>E</td>
<td>106</td>
<td>j</td>
<td>107</td>
<td>k</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>acknowledgement</td>
<td>38</td>
<td>&amp;</td>
<td>70</td>
<td>F</td>
<td>108</td>
<td>l</td>
<td>109</td>
<td>m</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>bell</td>
<td>39</td>
<td></td>
<td>71</td>
<td>G</td>
<td>110</td>
<td>n</td>
<td>111</td>
<td>o</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>back Space</td>
<td>40</td>
<td>(</td>
<td>72</td>
<td>H</td>
<td>112</td>
<td>p</td>
<td>113</td>
<td>q</td>
</tr>
<tr>
<td>9</td>
<td>HT</td>
<td>Horizontal Tab</td>
<td>41</td>
<td>)</td>
<td>73</td>
<td>I</td>
<td>114</td>
<td>r</td>
<td>115</td>
<td>s</td>
</tr>
<tr>
<td>10</td>
<td>LF</td>
<td>Line Feed</td>
<td>42</td>
<td>*</td>
<td>74</td>
<td>J</td>
<td>116</td>
<td>t</td>
<td>117</td>
<td>u</td>
</tr>
<tr>
<td>11</td>
<td>VT</td>
<td>Vertical Tab</td>
<td>43</td>
<td>+</td>
<td>75</td>
<td>K</td>
<td>118</td>
<td>v</td>
<td>119</td>
<td>w</td>
</tr>
<tr>
<td>12</td>
<td>FF</td>
<td>Form Feed</td>
<td>44</td>
<td>-</td>
<td>76</td>
<td>L</td>
<td>120</td>
<td>x</td>
<td>121</td>
<td>y</td>
</tr>
<tr>
<td>13</td>
<td>CR</td>
<td>Carriage Return</td>
<td>45</td>
<td>.</td>
<td>77</td>
<td>M</td>
<td>122</td>
<td>z</td>
<td>123</td>
<td>{</td>
</tr>
<tr>
<td>14</td>
<td>SO</td>
<td>Shift Out</td>
<td>46</td>
<td></td>
<td>78</td>
<td>N</td>
<td>124</td>
<td>}</td>
<td>125</td>
<td>[</td>
</tr>
<tr>
<td>15</td>
<td>SI</td>
<td>Shift In</td>
<td>47</td>
<td>/</td>
<td>79</td>
<td>O</td>
<td>126</td>
<td>\</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>DLE</td>
<td>Data Link Escape</td>
<td>48</td>
<td>0</td>
<td>80</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>DC1</td>
<td>Device Control 1</td>
<td>49</td>
<td>1</td>
<td>81</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>DC2</td>
<td>Device Control 2</td>
<td>50</td>
<td>2</td>
<td>82</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>DC3</td>
<td>Device Control 3</td>
<td>51</td>
<td>3</td>
<td>83</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>DC4</td>
<td>Device Control 4</td>
<td>52</td>
<td>4</td>
<td>84</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>NAK</td>
<td>Negative Acknowledgement</td>
<td>53</td>
<td>5</td>
<td>85</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>SYN</td>
<td>Synchronous Idle</td>
<td>54</td>
<td>6</td>
<td>86</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>ETB</td>
<td>End of Trans Block</td>
<td>55</td>
<td>7</td>
<td>87</td>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>CAN</td>
<td>Cancel</td>
<td>56</td>
<td>8</td>
<td>88</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>EM</td>
<td>End of Medium</td>
<td>57</td>
<td>9</td>
<td>89</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>SUB</td>
<td>Substitute</td>
<td>58</td>
<td>;</td>
<td>90</td>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>ESC</td>
<td>Escape</td>
<td>59</td>
<td></td>
<td>91</td>
<td>[</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>FS</td>
<td>File Separator</td>
<td>60</td>
<td>&lt;</td>
<td>92</td>
<td>\</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>GS</td>
<td>Group Separator</td>
<td>61</td>
<td>=</td>
<td>93</td>
<td>]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>RS</td>
<td>Record Separator</td>
<td>62</td>
<td>&gt;</td>
<td>94</td>
<td>^</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>US</td>
<td>Unit Separator</td>
<td>63</td>
<td>?</td>
<td>95</td>
<td>_</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C Program to Print Character Set

```c
#include<stdio.h>
#include<conio.h>

int main() {
    int i; clrscr();
    printf("ASCII ==> Character\n");
    for(i = -128; i <= 127; i++)
        printf("%d ==> %c\n", i, i);
    return 0;
}
```
Float and Double Point Numbers - IEEE 754 standard

**Single Precision**

- **s** (sign bit)
- **exp** (exponent)
- **mantissa**

32 bits in total, with 23 bits for the mantissa.

**Double Precision**

- **s** (sign bit)
- **exp** (exponent)
- **mantissa**

64 bits in total, with 52 bits for the mantissa.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Size</th>
<th>Range</th>
<th>Decimal Places</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>4 byte</td>
<td>1.2E-38 to 3.4E+38</td>
<td>6</td>
</tr>
<tr>
<td>double</td>
<td>8 byte</td>
<td>2.3E-308 to 1.7E+308</td>
<td>15</td>
</tr>
<tr>
<td>long double</td>
<td>10 byte</td>
<td>3.4E-4932 to 1.1E+4932</td>
<td>19</td>
</tr>
</tbody>
</table>
What is the decimal value of this Single Precision float?

Solution:
- Sign = 1 is negative
- Exponent = (01111100)₂ = 124, $E - bias = 124 - 127 = -3$
- Significand = $(1.0100 \ldots 0)_2 = 1 + 2^{-2} = 1.25$ (1 is implicit)
- Value in decimal = $-1.25 \times 2^{-3} = -0.15625$

What is the decimal value of?

Solution:
- Value in decimal = $+(1.01001100 \ldots 0)_2 \times 2^{130-127} = (1.01001100 \ldots 0)_2 \times 2^3 = (1010.01100 \ldots 0)_2 = 10.375$
If you know the abstraction you can go In and modify anything!
C Structures

• A powerful tool for developing your own data abstractions

```c
struct structNameName {
    type name;
    ..... 
};
```
What Abstractions for a Finite Element Method Application?

- Node
- Element
- Load
- Constraint
- Domain
- Vector
- Matrix
What is in a Node?

• Node number or tag
• Coordinates
• Displacements?
• Velocities and Accelerations??

2d or 3d?
How many dof?
Do We Store Velocities and Accel.

 Depends on what the program needs of it
Say Requirement is 2dimensional, need to store the displacements (3dof)?

```c
struct node {
    int tag;
    double xCrd;
    double yCrd;
    double displX;
    double dispY;
    double rotZ;
};
```

```c
struct node {
    int tag;
    double coord[2];
    double displ[3];
};
```

I would lean towards the latter; easier to extend to 3d w/o changing 2d code, easy to write for loops .. But is there a cost associated with accessing arrays instead of variable directly .. Maybe compile some code and time it for intended system
#include <stdio.h>

struct node {
    int tag;
    double coord[2];
    double disp[3];
};

void nodePrint(struct node *);

int main(int argc, const char **argv) {
    struct node n1; // create variable named n1 of type node
    struct node n2;
    n1.tag = 1; // to set n1’s tag to 1 .. Notice the DOT notation
    n1.coord[0] = 0.0;
    n1.coord[0] = 1.0;
    n2.tag = 2;
    n2.coord[0] = n1.coord[0];
    n2.coord[0] = 2.0;
    nodePrint(&n1);
    nodePrint(&n2);
}

void nodePrint(struct node *theNode) {
    printf("Node : %d ", theNode->tag);  // because the object is a pointer use -> ARROW to access
    printf("Crds: %f %f ", theNode->coord[0], theNode->coord[1]);
    printf("Disp: %f %f %f 
", theNode->disp[0], theNode->disp[1], theNode->disp[2]);
}
#include <stdio.h>

typedef struct node {
    int tag;
    double coord[2];
    double disp[3];
} Node;

void nodePrint(Node *);
void nodeSetup(Node *, int tag, double crd1, double crd2);

int main(int argc, const char **argv) {
    Node n1;
    Node n2;
    nodeSetup(&n1, 1, 0., 1.);
    nodeSetup(&n2, 2, 0., 2.);
    nodePrint(&n1);
    nodePrint(&n2);
}

void nodePrint(Node *theNode){
    printf("Node : %d ", theNode->tag);
    printf("Crds: %f %f 
", theNode->coord[0], theNode->coord[1]);
    printf("Disp: %f %f %f \n", theNode->disp[0], theNode->disp[1], theNode->disp[2]);
}

void nodeSetup(Node *theNode, int tag, double crd1, double crd2) {
    theNode->tag = tag;
    theNode->coord[0] = crd1;
    theNode->coord[1] = crd2;
}
Clean This Up for Large Project:

- Files for each data type and the functions
  - node.h, node.c, domain.h, domain.c, ....

```c
#include "node.h"
#include "domain.h"
int main(int argc, const char **argv) {
    Domain theDomain;

    domainAddNode(&theDomain, 1, 0.0, 0.0);
    domainAddNode(&theDomain, 2, 0.0, 2.0);
    domainAddNode(&theDomain, 3, 1.0, 1.0);

    domainPrint(&theDomain);

    // get and print singular node
    printf("\nsingular node:\n");
    Node *theNode = domainGetNode(&theDomain, 2);
    nodePrint(theNode);
}
```
Domain

- Container class to store nodes, elements, loads, constraints, ...
- What storage scheme for the different data types?
- What are the options:
  - Array
  - Linked List
  - Double Linked List
  - Tree
  - Hybrid approaches ......
```c
#ifndef _DOMAIN
#define _DOMAIN

#include "node.h"

typedef struct struct_domain {
    Node *theNodes;
} Domain;

void domainPrint(Domain *theDomain);
void domainAddNode(Domain *theDomain, int tag, double crd1, double crd2);
void domainPrintNodes(Domain *theDomain);
Node *domainGetNode(Domain *, int nodeTag);
#endif

#ifndef _NODE
#define _NODE

#include <stdio.h>

typedef struct node {
    int tag;
    double coord[2];
    double disp[3];
    struct node *next;
} Node;

void nodePrint(Node *);
void nodeSetup(Node *, int tag, double crd1, double crd2);
#endif
```
void domainAddNode(Domain *theDomain, int tag, double crd1, double crd2) {
    Node *theNextNode = (Node *)malloc(sizeof(Node));
    nodeSetup(theNextNode, tag, crd1, crd2);

    if (theDomain->theNodes != NULL) {
        theNextNode->next = theDomain->theNodes;
    }
    theDomain->theNodes = theNextNode;
}

void domainPrintNodes(Domain *theDomain) {
    Node *theCurrentNode = theDomain->theNodes;
    while (theCurrentNode != NULL) {
        nodePrint(theCurrentNode);
        theCurrentNode = theCurrentNode->next;
    }
}

Node *domainGetNode(Domain *theDomain, int nodeTag) {
    Node *theCurrentNode = theDomain->theNodes;
    while (theCurrentNode != NULL) {
        if (theCurrentNode->tag == nodeTag) {
            return theCurrentNode;
        } else theCurrentNode = theCurrentNode->next;
    }
    return NULL;
}
Object Oriented Programming and C++
How do We Now Add Elements to the FEM code?

- Want 2d beam elements

```c
typedef struct struct_domain {
    Node *theNodes;
    Constraints *theConstraints;
    Beam *theBeams
} struct_domain;
```

And Trusses!

```c
typedef struct struct_domain {
    Node *theNodes;
    Constraints *theConstraints;
    Beam *theBeams;
    Truss *theTrusses;
} struct_domain;
```

Why Not Just Elements .. That requires some functional pointers!
Problem With C is Certain Data & Functions Separate so need these function pointers

Object-Oriented Programming Offers a Solution
Object-Oriented Programming overcomes the problem by something called *encapsulation*. The data and functions (methods) are bundled together into a class. The class presents an interface, hiding the data and implementation details. If written correctly only the class can modify the data. The functions or other classes in the program can only query the methods, the interface functions.
Object-Oriented Programs all provide the ability of one class to inherit the behaviour of a parent class (or even multiple parent classes). This allows the Beam and Trusses both to be treated just as elements. The are said to be polymorphic.
C++

• Developed by Bjourne Stroustrup working at Bell Labs (again) in 1979. Originally “C With Classes” it was renamed C++ in 1983.

• A general purpose programming language providing both functional and object-oriented features.

• As an incremental upgrade to C, it is both strongly typed and a compiled language.

• The updates include:
  • Object-Oriented Capabilities
  • Standard Template Libraries
  • Additional Features to make C Programming easier!
C++ Program Structure

A C++ Program consists of the following parts:

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments
- Classes
Hello World (of course C Hello World also works!)

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

int main() {
    /* my first program in C++ */
    cout << "Hello World! \n";
    return 0;
}
```
#include <iostream>
using namespace std;

int main(int argc, char **argv) {
    int n;
    double *array1=0, *array2=0, *array3=0;

    // get n
    cout << "enter n: ";
    cin >> n;
    if (n <=0) {printf ("You idiot\n"); return(0);}

    // allocate memory & set the data
    array1 = new double[n];
    for (int i=0; i<n; i++) {
        array1[i] = 0.5*i;
    }
    array2 = array1;
    array3 = &array1[0];

    for (int i=0; i<n; i++, array3++) {
        double value1 = array1[i];
        double value2 = *array2++;
        double value3 = *array3;
        printf("%.4f %.4f %.4f\n", value1, value2, value3);
    }

    // free the array
    delete array1[];
    return(0);
}

You should not malloc something and delete it later or new something and free it later. The bahviour is undefined. “The program may continue normally, it may crash immediately, it may produce a well-defined error message and exit gracefully, it may start exhibiting random errors at some time after the actual undefined behavior event”

```
c >gcc memory1.c; ./a.out
enter n: 5
 0.0000  0.0000  0.0000
 0.5000  0.5000  0.5000
 1.0000  1.0000  1.0000
 1.5000  1.5000  1.5000
 2.0000  2.0000  2.0000

c >./a.out
enter n: 3
 0.0000  0.0000  0.0000
 0.5000  0.5000  0.5000
 1.0000  1.0000  1.0000
```

c >
#include <iostream>
#include <string>
using namespace std;

int main(int argv, char **argv) {
  string pName = argv[0];
  string str;
  cout << "Enter Name: ";
  cin >> str;

  if (pName == "./a.out")
    str += " the lazy sod";

  str += " says ";
  str = str + "HELLO World";
  cout << str << "\n";
  return 0;
}
#include <iostream>
using namespace std;

void sum1(int a, int b, int *c);
void sum2(int a, int b, int &c);

int main(int argc, char **argv) {
    int x = 10;
    int y = 20;
    int z;
    sum1(x, y, &z);
    cout << x << " + " << y << " = " << z << "\n";
    x = 20;
    sum2(x, y, z);
    cout << x << " + " << y << " = " << z << "\n";
}

// c by value
void sum1(int a, int b, int *c) {
    *c = a + b;
}

// c by ref
void sum2(int a, int b, int &c) {
    c = a + b;
}
Classes

A class in C++ is the programming code that defines the methods (defines the api) in the class interface and the code that implements the methods. For classes to be used by other classes and in other programs, these classes will have the interface in a .h file and the implementation in a .cpp (.cc, .".cxx", or ".c++”) file.
class Shape {
public:
    virtual ~Shape();
    virtual double GetArea(void) = 0;
    virtual void PrintArea(ostream &s);
};

• keywod class defines this as a class, Shape is the name of the class
• Classes can have 3 sections:
  1. Public: objects of all other classes and program functions can invoke this method on the object
  2. Protected: only objects of subclasses of this class can invoke this method.
  3. Private: only objects of this specific class can invoke the method.

• virtual double GetArea(void) = 0 , the =0; makes this an abstract class. (It cannot be instantiated.) It says the class does not provide code for this method. A subclass must provide the implementation.

• virtual void PrintArea(ostream &s) the class provides an implementation of the method, the virtual a subclass may also provide an implementation.

• virtual ~Shape() is the destructor. This is method called when the object goes away either through a delete or falling out of scope.
class Shape {
public:
    virtual ~Shape();
    virtual double GetArea(void) = 0;
    virtual void PrintArea(ostream &s);
};

class Rectangle: public Shape {
public:
    Rectangle(double w, double h);
    ~Rectangle();
    double GetArea(void);
    void PrintArea(ostream &s);
protected:
    // shared by subclasses
private:
    double width, height;
    static int numRect;
};

• **class Rectangle**: **public Shape** defines this as a class, **Rectangle** which is a subclass of the class **Shape**.
• It has 3 sections, public, protected, and private.
• It has a constructor **Rectangle(double w, double h)** which states that class takes 2 args, w and h when creating an object of this type.
• It also provides the methods **double GetArea(void)** and **void PrintArea(ostream &s)**; Neither are virtual which means no subclass can provide an implementation of these methods.
• In the private area, the class has 3 variables. Width and height are unique to each object and are not shared. Numrect is shared amongst all objects of type Rectangle.
• **class Circle: public Shape** defines this as a class **Circle** which is a subclass of the class Shape.

  • It has 2 sections, public and private.

  • It has a constructor **Circle(double d)** which states that class takes 1 arg d when creating an object of this type.

  • It also provides the method **double GetArea(void)**.

  • **There is no PrintArea() method, meaning this class relies on the base class implementation.**

  • In the private area, the class has 1 variable and defines a private method, GetPI(). Only objects of type Circle can invoke this method.
Shape::~Shape() {
    cout << "Shape Destructor\n";
}

void Shape::PrintArea(ostream &s) {
    s << "UNKOWN area: " << this->GetArea() << "\n";
}
- `int Rectangle::numRect = 0` creates the memory location for the classes static variable `numRect`.
- The `Rectangle::Rectangle(double w, double d)` is the class constructor taking 2 args.
- The line `:Shape(), width(w), height(d)` is the first code exe. It calls the base class constructor and then sets it’s 2 private variables.
- The constructor also increments the static variable in `numRect++;` That variable is decremented in the destructor.
Circle::~Circle() {
    cout << "Shape Destructor\n";
}

Circle::Circle(double d) {
    diameter = d;
}

double Circle::GetArea(void) {
    return this->GetPI() * diameter * diameter/4.0;
}

Double Circle::GetPI(void) {
    return 3.14159;
}

• Last but not least!
```cpp
#include <iostream>
#include "shape1.h"

using namespace std;

int main(int argc, char **argv) {
    Circle s1(2.0);
    Shape *s2 = new Rectangle(1.0, 2.0);
    Shape *s3 = new Rectangle(3.0, 2.0);

    s1.PrintArea(cout);
    s2->PrintArea(cout);
    s3->PrintArea(cout);

    return 0;
}
```

When we run it, results should be as you expected. Notice the destructors for s2 and s3 objects not called. The delete was not invoked. Also notice order of destructor calls, base class destructed last.

s1 is a variable of type Circle. To invoke methods on this object we use the DOT.

s2 and s3 are pointers to objects created with new. To invoke methods on these objects from our pointer variables we use the ARROW ->
Containers

- Order is important?
  - Yes
    - First In First Out
      - Yes
        - Larger First Element Out
          - Yes
            - Priority Queue
          - No
            - Sorted by key
            - No
            - Insert/erase in middle
              - Yes
                - Insert/erase at front
                  - Yes
                    - Store key separately by value
                      - Yes
                        - Multi_map
                      - No
                        - Multi_set
                    - No
                      - Size will vary widely
                        - Yes
                          - Need merge collections
                            - Yes
                              - List
                            - No
                              - Vector
                        - No
                          - Need to merge collections
                            - Yes
                              - Set
                            - No
                              - Map
  - No
    - Last In First Out
      - Yes
        - Need to find element by key?
          - Yes
            - Stack
          - No
            - Queue
    - No
      - Insert/erase in middle?
        - Yes
          - Need to find the n-th element
            - Yes
              - Size will vary widely
                - Yes
                  - Need merge collections
                    - Yes
                      - Deque
                    - No
                      - Vector
                - No
                  - Map
            - No
              - Need to merge collections
                - Yes
                  - List
                - No
                  - Set
        - No
          - Insert/erase at front
            - Yes
              - Store key separately by value
                - Yes
                  - Multi_map
                - No
                  - Multi_set
            - No
              - Sort by key
                - Yes
                  - Priority Queue
                - No
                  - Stack
Domain.h

```cpp
#ifndef _DOMAIN
#define _DOMAIN

#include "Domain.h"
#include <map>
class Node;
using namespace std;

class Domain {
public:
    Domain();
    ~Domain();

    Node *getNode(int tag);
    void Print(ostream &s);
    int AddNode(Node *theNode);

private:
    map<int, Node *>theNodes;
};
#endif
```

- The `#ifndef`, `#define`, `#endif` are important to put in every header file to potentially stop compiler going into an infinite loop.

- To store the nodes we are using a built in STL container of type `map<int, Node *>`;
Domain.cpp

Node *
Domain::getNode(int tag) {
    Node *res = NULL;

    // create iterator & iterate over all elements
    std::map<int, Node *>::iterator it = theNodes.begin();

    while (it != theNodes.end()) {
        Node *theNode = it->second;
        if (theNode->GetTag() == tag) {
            res = theNode;
            break;
        }
        it++;
    }

    return res;
}

We create an iterator for our particular map. Then we simply iterate until we either find the node we want or we reach the end of the elements in the map.

Syntax is bloody awful, but they are very powerful.