Modifying objects

Operators and Expressions

Memory Depiction

```java
float y = 12.5;
```
Memory Depiction

```
float y = 12.5;
int Temperature = 32;
char Letter = 'c';
```

Memory Depiction

```
float y = 12.5;
int Temperature = 32;
char Letter = 'c';
```
Memory Depiction

```plaintext
float y = 12.5;
int Temperature = 32;
char Letter = 'c';
int Number;

12.5 32 'c'
y Temperature Letter Number
1001 1002 1003 1007 1008 1009
```

Assignment Statement

- **Basic form**
  - `object = expression;`
  - `Celsius = (Fahrenheit - 32) * 5 / 9;`
  - `y = m * x + b;`

- **Action**
  - Expression is evaluated
  - Expression value stored in object
Definition

```c
int NewStudents = 6;

int OldStudents = 21;
```

<table>
<thead>
<tr>
<th>NewStudents</th>
<th>OldStudents</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>21</td>
</tr>
</tbody>
</table>
Definition

int NewStudents = 6;
int OldStudents = 21;
int TotalStudents;

<table>
<thead>
<tr>
<th>NewStudents</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>OldStudents</td>
<td>21</td>
</tr>
<tr>
<td>TotalStudents</td>
<td>?</td>
</tr>
</tbody>
</table>

Assignment Statement

int NewStudents = 6;
int OldStudents = 21;
int TotalStudents;

TotalStudents = NewStudents + OldStudents;
Assignment Statement

int NewStudents = 6;
int OldStudents = 21;
int TotalStudents;

TotalStudents = NewStudents + OldStudents;

<table>
<thead>
<tr>
<th>NewStudents</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>OldStudents</td>
<td>21</td>
</tr>
<tr>
<td>TotalStudents</td>
<td>27</td>
</tr>
</tbody>
</table>
Assignment Statement

```java
int NewStudents = 6;
int OldStudents = 21;
int TotalStudents;

TotalStudents = NewStudents + OldStudents;

OldStudents = TotalStudents;
```

Consider

```java
int Value1 = 10;
```
Consider

\[
\begin{align*}
\text{int Value1 = 10;} \\
\text{int Value2 = 20;}
\end{align*}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Value1} & 10 \\
\hline
\text{Value2} & 20 \\
\hline
\end{array}
\]

Consider

\[
\begin{align*}
\text{int Value1 = 10;} \\
\text{int Value2 = 20;} \\
\text{int Hold = Value1;}
\end{align*}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Value1} & 10 \\
\hline
\text{Value2} & 20 \\
\text{Hold} & 10 \\
\hline
\end{array}
\]
Consider

```c
int Value1 = 10;
int Value2 = 20;
int Hold = Value1;

Value1 = Value2;
```

Consider

```c
int Value1 = 10;
int Value2 = 20;
int Hold = Value1;

Value1 = Value2;
```
Consider

int Value1 = 10;
int Value2 = 20;
int Hold = Value1;

Value1 = Value2;
Value2 = Hold;

We swapped the values of objects Value1 and Value2 using Hold as temporary holder for Value1’s starting value!
Incrementing

```java
int i = 1;
```

Assign the value of expression `i + 1` to `i`

Evaluates to 2
Const Definitions

- Modifier `const` indicates that an object cannot be changed
  - Object is read-only
- Useful when defining objects representing physical and mathematical constants
  ```cpp
  const float Pi = 3.1415;
  ```
- Value has a name that can be used throughout the program
  ```cpp
  const int SampleSize = 100;
  ```
- Makes changing the constant easy
  - Only need to change the definition and recompile

Assignment Conversions

- Floating-point expression assigned to an integer object is truncated
- Integer expression assigned to a floating-point object is converted to a floating-point value
- Consider
  ```cpp
  float y = 2.7;
  int i = 15;
  int j = 10;
  i = y;              // i is now 2
  cout << i << endl;
  y = j;              // y is now 10.0
  cout << y << endl;
  ```
Nonfundamental Types

- Nonfundamental as they are additions to the language
- C++ permits definition of new types and classes
  - A class is a special kind of type
- Class objects typically have
  - Data members that represent attributes and values
  - Member functions for object inspection and manipulation
  - Members are accessed using the selection operator (.)
    \[ j = s.\text{size}(); \]
  - Auxiliary functions for other behaviors
- Libraries often provide special-purpose types and classes
- Programmers can also define their own types and classes

Examples

- Standard Template Library (STL) provides class string
- EzWindows library provides several graphical types and classes
  - SimpleWindow is a class for creating and manipulating window objects
  - RectangleShape is a class for creating and manipulating rectangle objects
Class string

- Class string
  - Used to represent a sequence of characters as a single object
- Some definitions
  ```
  string Name = "Joanne";
  string DecimalPoint = ".";
  string empty = ";
  string copy = name;
  string Question = '?'; // illegal
  ```

Nonfundamental Types

- To access a library use a preprocessor directive to add its definitions to your program file
  ```
  #include <string>
  ```
- The using statement makes syntax less clumsy
  - Without it
    ```
    std::string s = "Sharp";
    std::string t = "Spiffy";
    ```
  - With it
    ```
    using namespace std; // std contains string
    string s = "Sharp";
    string t = "Spiffy";
    ```
EzWindows Library Objects

Definitions are the same form as other objects
Example

```c
SimpleWindow W;
```
- Most non-fundamental classes have been created so that an object is automatically initialized to a sensible value

SimpleWindow objects have member functions to process messages to manipulate the objects
- Most important member function is `Open()` which causes the object to be displayed on the screen
  Example
  ```c
  W.Open();
  ```

Initialization

Class objects may have several attributes to initialize

Syntax for initializing an object with multiple attributes
```
Type Identifier(Exp1, Exp2, ..., Expn);
```

SimpleWindow object has several optional attributes
```
SimpleWindow W("Window Fun", 8, 4);
```
- First attribute
  - Window banner
- Second attribute
  - Width of window in centimeters
- Third attribute
  - Height of window in centimeters
# An EzWindows Program

```cpp
#include <iostream>
using namespace std;
#include "ezwin.h"
int ApiMain() {
    SimpleWindow W("A Window", 12, 12);
    W.Open();

    cout << "Enter a character to exit" << endl;
    char a;
    cin >> a;

    return 0;
}
```

# An EzWindows Project File
An EzWindows Project File

Sample Display Behavior
RectangleShape Objects

- EzWindows also provides RectangleShape for manipulating rectangles.

- RectangleShape objects can specify the following attributes:
  - SimpleWindow object that contains the rectangle (mandatory)
  - Offset from left edge of the SimpleWindow
  - Offset from top edge of the SimpleWindow
    - Offsets are measured in centimeters from rectangle center
  - Width in centimeters
  - Height in centimeters
  - Color
    - color is an EzWindows type

Examples

SimpleWindow W1("My Window", 20, 20);
SimpleWindow W2("My Other Window", 15, 10);

RectangleShape R(W1, 4, 2, Blue, 3, 2);
RectangleShape S(W2, 5, 2, Red, 1, 1);
RectangleShape T(W1, 3, 1, Black, 4, 5);
RectangleShape U(W1, 4, 9);
RectangleShape Objects

- Some RectangleShape member functions for processing messages
  - **Draw()**
    - Causes rectangle to be displayed in its associated window
  - **GetWidth()**
    - Returns width of object in centimeters
  - **GetHeight()**
    - Returns height of object in centimeters
  - **SetSize()**
    - Takes two attributes -- a width and height -- that are used to reset dimensions of the rectangle

Another EzWindows Program

```cpp
#include <iostream>
using namespace std;
#include "rect.h"
int ApiMain() {
    SimpleWindow W("Rectangular Fun", 12, 12);
    W.Open();
    RectangleShape R(W, 5.0, 2.5, Blue, 1, 2);
    R.Draw();
    cout << "Enter a character to exit" << endl;
    char Response;
    cin >> Response;
    return 0;
}
```
Sample Display Behavior

C++ has a large set of operators for applying an operation to an object and then storing the result back into the object.

Examples

```cpp
int i = 3;
i += 4; // i is now 7
cout << i << endl;

float a = 3.2;
a *= 2.0; // a is now 6.4
cout << a << endl;
```
Increment and Decrement

- C++ has special operators for incrementing or decrementing an object by one
- Examples
  ```c++
  int k = 4;
  ++k;       // k is 5
  k++;       // k is 6
  cout << k << endl;
  int i = k++;
  cout << i << " " << k << endl;  // i is 6, k is 7
  int j = ++k;
  cout << j << " " << k << endl;  // j is 8, k is 8
  ```

Class string

- Some string member functions
  - size() determines number of characters in the string
    ```c++
    string Saying = "Rambling with Gambling";
    cout << Saying.size() << endl;       // 22
    ```
  - substr() determines a substring (Note first position has index 0)
    ```c++
    string Word = Saying.substr(9, 4);   // with
    ```
  - find() computes the position of a subsequence
    ```c++
    int j = Saying.find("it");          // 10
    int k = Saying.find("its");         // ?
    ```
Class string

Auxiliary functions and operators

- getline() extracts the next input line

```cpp
string Response;
cout << "Enter text: ";
generateLine(cin, Response, '\n');
cout << "Response is \"" << Response
    << "\"" << endl;
```

- Example run

```
Enter text: Want what you do
Response is "Want what you do"
```

Class string

Auxiliary operators

- + string concatenation

```cpp
string Part1 = "Me";
string Part2 = " and ";
string Part3 = "You";
string All = Part1 + Part2 + Part3;
```

- += compound concatenation assignment

```cpp
string ThePlace = "Brooklyn";
ThePlace += ", NY";
```
```cpp
#include <iostream>
using namespace std;

int main() {
    cout << "Enter the date in American format: " << "(e.g., January 1, 2001) : ";
    string Date;
    getline(cin, Date, '\n');
    int i = Date.find(" ");
    string Month = Date.substr(0, i);
    int k = Date.find(",");
    string Day = Date.substr(i + 1, k - i - 1);
    string Year = Date.substr(k + 2, Date.size() - 1);
    string NewDate = Day + " " + Month + " " + Year;
    cout << "Original date: " << Date << endl;
    cout << "Converted date: " << NewDate << endl;
    return 0;
}
```