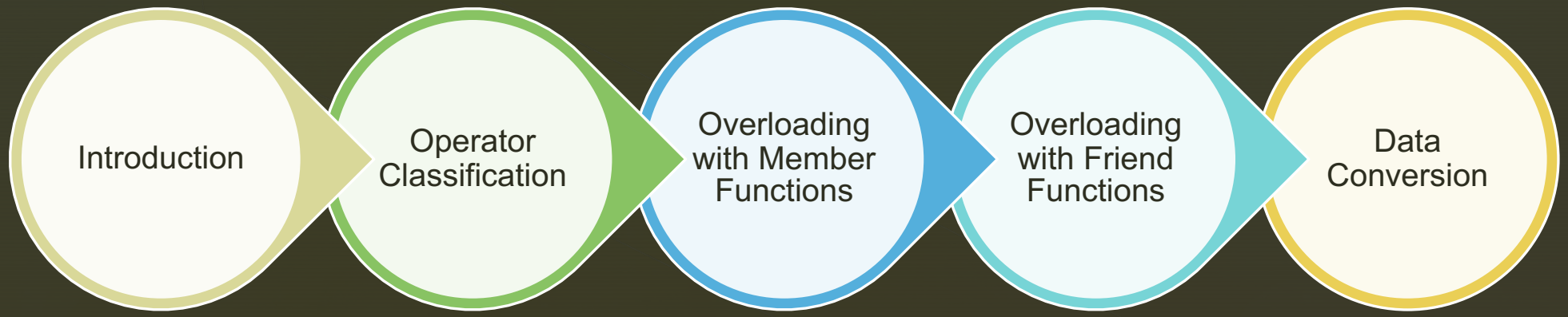
DR. (MRS) T-S.M.A. ADJAIDOO

10

Operator Overloading

Today's Lesson



Learning Outcomes

1
- Learners must understand the need to overload

2
- Learners must know how to convert data

3
- Learners must know how to overload functions

Operator Overloading

Introduction

DEFINITION

- *OPERATOR OVERLOADING* refers to giving the normal C++ operators, such as `+`, `*`, `<=`, and `+=`, additional meanings when they are applied to user-defined data types(classes).
- Operator overloading is the ability to tell the compiler how to perform a certain operation when its corresponding operator is used on one or more variables.
- C++ operators have specific defined meanings relative to built-in types.

C++ Operators

Operators	Meaning
+ - * / % ++ --	Arithmetic Operators
== != < <= > >=	Relational Operators
&& || !	Logical Operators
=	Assignment Operators
& | ^ ~ << >>	Bitwise Operators
() []	Function call, subscript operators
& * -> , new delete	Other Operators
op=	Binary arithmetic or binary bitwise operator

Introduction

- These operators can be redefined relative to user-defined types.
- An operator is always overloaded in conjunction with a class.
- The definition scope of an operator is simply extended—the characteristics of the operator remain unchanged.

Introduction

RULES OF OPERATOR OVERLOADING

- The individual original properties of the operators are maintained.
- You cannot create "new operators"—that is, you can only overload existing operators.
- You cannot change the arity of an operator. A binary operator will always be binary and a unary operator will always be unary.
- The precedence and the order of grouping operators of the same precedence remains unchanged.

Introduction

- For example, a stack class may use ‘+’ to push items onto the stack, ie. `A+8` pushes ‘`8`’ onto the stack object ‘`A`’.
- To overload an operator, its meaning must be defined relative to the class to which it is applied.
- This is done with the operator function;

```
type classname::operator#(arg list){
    operation relative to class
    }
```

Introduction

- Operator overloading allows easy integration of new data types in a program.
- Vector objects `A(1, 2, 3)` and `B(10, 10, 10)` can be easily added using an overloaded '+' to get `C(1+10, 2+10, 3+10)`.
- Note however that, consistency must be maintained.

Operator Classification

- Binary and unary operators are overloadable.
- Examples; binary: `-, +, %, =, <=, /, *,` etc.

  unary: `!, - , +, ++, --.`
- Ternary/conditional operator(`?:`), dot operator(.),pointer-to-member operator(=>) and the scope resolution operator(`::`) are not overloadable.

Operator Classification

- Operator functions can be member or non-member functions.
- Overloading operator functions of assignment operators ie. `()`, `[]`, `=`, is only possible if it is a member function.
- A non-member function must be a `friend` if `private` or `protected` members of that class are accessed directly.

Overloading With Member Functions

```
// Overload operators using member functions.
#include <iostream>
using namespace std;

class three_d {
    int x, y, z; // 3-D coordinates
    public:
        three_d()
            { x = y = z = 0; }
        three_d(int i, int j, int k)
            {x = i; y = j; z = k; }
        three_d operator +(three_d op2); // op1 is implied
        three_d operator =(three_d op2); // op1 is implied
        three_d operator ++();
        three_d operator ++(int);

        void show();
};
```

Overloading With Member Functions

```
// Overload +.
three_d three_d::operator +(three_d op2){
    three_d temp;
    temp.x = x + op2.x; // These are integer additions
    temp.y = y + op2.y; // and the + retains its original
    temp.z = z + op2.z; // meaning relative to them.
    return temp;
}
// Overload assignment.
three_d three_d::operator =(three_d op2){
    x = op2.x; // These are integer assignments
    y = op2.y; // and the = retains its original
    z = op2.z; // meaning relative to them.
    return *this;
}
```

Overloading With Member Functions

```
// Overload the postfix version of ++.
three_d three_d::operator ++(int){
    three_d temp = *this; // save original value
    x++;  y++;  z++;   // increment x, y, and z
    return temp; // return original value
}
// Overload the prefix version of ++.
three_d three_d::operator ++(){
    x++;  y++;  z++;   // increment x, y, and z
    return *this; // return altered value
}
// Show X, Y, Z coordinates.
void three_d::show(){
    cout << x << ", ";
    cout << y << ", ";
    cout << z << "\n";
}
```

Overloading With Member Functions

Output

```
1, 2, 3
10, 10, 10
11, 12, 13
22, 24, 26
2 1, 2, 3
2, 3, 4
3, 4, 5
3 4, 5, 6
5, 6, 7
```

```
int main()
{
    three_d a(1, 2, 3), b(10, 10, 10), c;
    a.show();  //1
    b.show();  //2
    c = a + b; // add a and b together
    c.show();  //3
    c = a + b + c; // add a, b and c together
    c.show();  //4
    c = b = a; // demonstrate multiple assignment
    c.show();  //5
    b.show();  //6
    ++c; // prefix increment
    c.show();  //7
    c++; // postfix increment
    c.show();  //8
    a = ++c; // a receives c's value after increment
    a.show();  //9
    c.show();  //10
    a = c++; // a receives c's value prior to increment
    a.show();  //11
    c.show();  //12
    return 0;
}
```

Overloading With Member Functions

- Note that binary operator member functions take only one argument.
- The operand to the left of the binary operator calls the function and passes the other operand through the argument.
- Parameters of the calling operand are passed through the in-built `this` pointer.

Overloading With Member Functions

- Unary operator member functions take no argument.
- The arguments of the only operand are passed through the `this` pointer.
- The parameter `int` is not used by the function, and should be ignored. This parameter is simply a way for the compiler to distinguish between the prefix and postfix forms of the increment operator.

Overloading With `friend` Functions

- Overloading binary operators using `friend` functions take two arguments.
- Unary operators take one argument, in this case but must be avoided as much as possible.
- Note that "`::`" is not used in the function definition.

Overloading With `friend` Functions

```
class three_d {
    int x, y, z; // 3-D coordinates
    public:
        three_d(){
            x = y = z = 0;
        }
        three_d(int i, int j, int k) {
            x = i; y = j; z = k;
        }
        friend three_d operator +(three_d op1, three_d op2);
        three_d operator =(three_d op2); // op1 is implied
        void show() ;
};

// This is now a friend function.
three_d operator +(three_d op1, three_d op2) {
    three_d temp;
    temp.x = op1.x + op2.x;
    temp.y = op1.y + op2.y;
    temp.z = op1.z + op2.z;
    return temp;
}
```

Data Conversion

- The C++ compiler performs automatic conversions among fundamental data types.
- For example: assigning a float value to an integer object, the compiler stores a down-rounded value of the float.

Data Conversion

- Sometimes we use the cast operator to force the compiler to convert one type to another.
- To convert `float` to `int`, we can say;

```
intcal = static_cast<int>(floatcal);
```

- The compiler cannot do type conversions among user-defined types automatically.
- This can be achieved by programmer specification.

Data Conversion

- These are done using the conversion/cast operator.
- Instead of overloading operators multiple times with different argument types, we provide conversion operators to perform conversions.
- Overloaded operators can then perform type conversion to match their arguments.

Data Conversion

- Data conversion can be from:
  - Objects to Basic types
  - Basic types to Objects
  - Object to Object of another class

Pitfalls Of Operator Overloading And Conversion

- Even though operator overloading can make a program more understandable and readable, it could also make the program very difficult to understand when the operator is not overloaded in the right way.

Tips To Operator Overloading

USE SIMILAR MEANINGS

- Even it is possible to overload a ‘–’ sign in instances when addition is being done; it is advisable to use the ‘+’ sign in order to make the program more understandable.

USE SIMILAR SYNTAX

- For a statement like alpha += beta, alpha is set to the sum of alpha and beta hence the overloaded version should as much as possible do the same.

Tips To Operator Overloading

- Some syntactical characteristics of operators can however not be changed. Eg. A binary operator cannot be overloaded to a unary operator
- Reading and understanding a program becomes much more difficult if the number of overloaded operators becomes too many and they are not used in relation to their functions.

Tips To Operator Overloading

- Operators must therefore not be overloaded too often and must be overloaded in ways in which their meanings and functions will be easily understood.
- Avoid doing the same conversion in more than one way.

Any Questions?

The End

Contact: tsadjaidoo@knust.edu.gh

Office: Caesar Building, Room 413