CS010C

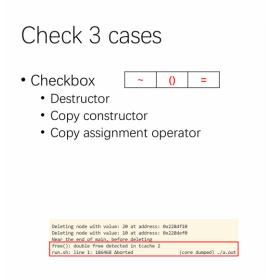
Lab4

Review: C++ class

- If you are using std funcitions, types, ...
 - Put using namespace std; at the top
 - Or add std:: every time you use them
- If you want to access a private variable from outside
 - Write an public accessor
 - Or temporarily change it to public(just for Lab2 demo)
- How to call a public member function from outside?
 - A Pointer of object
 - use -> (right arrow)
 - A Class object or a reference to a class object, etc.
 - use .(dot)

Lab2 demo: rule of 3

- Destructor ~
 - New dummy head & tail(not nullptr!)
 - Set their pointers
- Copy constructor ()
 - New dummy head & tail
 - For each node in "that" (or say "rhs") list, "push back" its value to "this" list
- Copy assignment operator =
 - Delete the nodes of "this" (or say "lhs") list
 - Then similar to copy constructor



list1: 0x7ffe27df0500 list1 : 0x7fffaf6af110 Dummy Head | address: 0x1b1aeb0 Dummy Head | address: 0x9f2eb0 value: 20 | address: 0x1b1af10 value: 20 | address: 0x9f2f10 value: 10 | address: 0x1b1aef0 value: 10 | address: 0x9f2ef0 Dummy Tail | address: 0x1b1aed0 Dummy Tail | address: 0x9f2ed0 Testing = operator list2: 0x7ffe27df04f0 list2 : 0x7fffaf6af100 Dummy Head | address: 0x9f2eb0 Dummy Head | address: 0x1b1b340 value: 20 | address: 0x1b1b3a0 value: 20 | address: 0x9f2f10 value: 10 | address: 0x1b1b380 value: 10 | address: 0x9f2ef0 Dummy Tail | address: 0x9f2ed0 Dummy Tail | address: 0x1b1b366 Testing copy constructor Testing copy constructor list3 : 0x7fffaf6af0f0 list3: 0x7ffe27df04e0 Dummy Head | address: 0x9f2eb0 Dummy Head | address: 0x1b1b3c6 value: 20 | address: 0x1b1b420 value: 20 | address: 0x9f2f10 value: 10 | address: 0x9f2ef0 value: 10 | address: 0x1b1b400 Dummy Tail | address: 0x9f2ed0 Dummy Tail | address: 0x1b1b3e6 Testing = at initialization Testing = at initialization list4 : 0x7fffaf6af0e0 list4: 0x7ffe27df04d0 Dummy Head | address: 0x9f2eb0 Dummy Head | address: 0x1b1b440 value: 20 | address: 0x9f2f10 value: 20 | address: 0x1b1b4a0 value: 10 | address: 0x9f2ef0 value: 10 | address: 0x1b1b480 Dummy Tail | address: 0x9f2ed0 Dummy Tail | address: 0x1b1b460 list5 : 0x9f3380 list5: 0x1b1b4c0 Dummy Head | address: 0x9f2eb0 Dummy Head | address: 0x1b1b4e0 value: 20 | address: 0x9f2f10 value: 20 | address: 0x1b1b540 value: 10 | address: 0x9f2ef0 value: 10 | address: 0x1b1b520 Dummy Tail | address: 0x9f2ed0 Dummy Tail | address: 0x1b1b500

Program2

stack

- Typedef: typedef int T;
 - Give int a new name T
 - Actually replace T with int

Concrete/Regular Function

- int main(int) argc, char const *argv[])
- "CertainType1 FunctionName(CertainType2 parameter1, ···)"
- a fixed return type and fixed parameter types
- Template Function
 - Return and parameter types are not fixed
 - Generate concrete functions for various specific types on demand
 - Multiple template types can be used for the parameters and return value

```
template <typename T, typename U>
TconvertTc(U) value)
{
    return static_cast<T>(value);
}
```

RPN(Postfix2Num)

- I/O hint
 - fstream

- std::string input = argv[1];
 std::ifstream file(input);
- negative numbers

- 77 (-6) + 2 / 10 5 3
- The first character is a '-'
- Not a single '-' (It is the minus symbol)

- token[0] == '-'
 token.size() > 1
- Idea: use a stack(algorithm available on zybooks)
 - Encounter a number
 - push
 - Encounter a symbol
 - Pop out 2 numbers from stack and perform the operation
 - Get an operation result. How should we handle it?

Infix2Postfix

- I/O hint
 - Fsteam: file >> token;
 - Works great for the most part, except for
 - '(' & ')'
 - Print the token you parse each time, to see if they were read correctly!
- Algorithm(see zybooks)
 - Precedence(low, '+' or '-')
 - Output & Pop until? _____high_low
 - Parenthesis(right, ')')
 - Output & Pop until a left parenthesis is encountered

```
(12 * (5 - 7) - 8 * (7 - 1)) / ((4 - 2) * 16)
```

```
Algorithm Infix-to-Postfix(infix expression P)
 // input: infix expression
 // output: postfix expression
 create an empty stack S to hold chars
 size = P.size()
  for each p[i] in P // where i = \{0, 1, ..., P.size()-1\}
   if p[i] is a operand
     output p[i]
    if p[i] is an operator: *, /, +, -
     output S.top() and S.pop() from S until left paren at S->top OR
     S.top() holds operator with lower precedence than p[i]
    if p[i] == '('
     S.push(p[i])
     // right paren causes stack to empty up to and including left paren
     S.top() and S.pop() to output from S until '(' found
     S.top() and S.pop() '(' to output
   while ( !S.empty )
     output S.top() // this is result
     S.pop()
     NOTE: Parentheses are not written to the output
```