CS010C

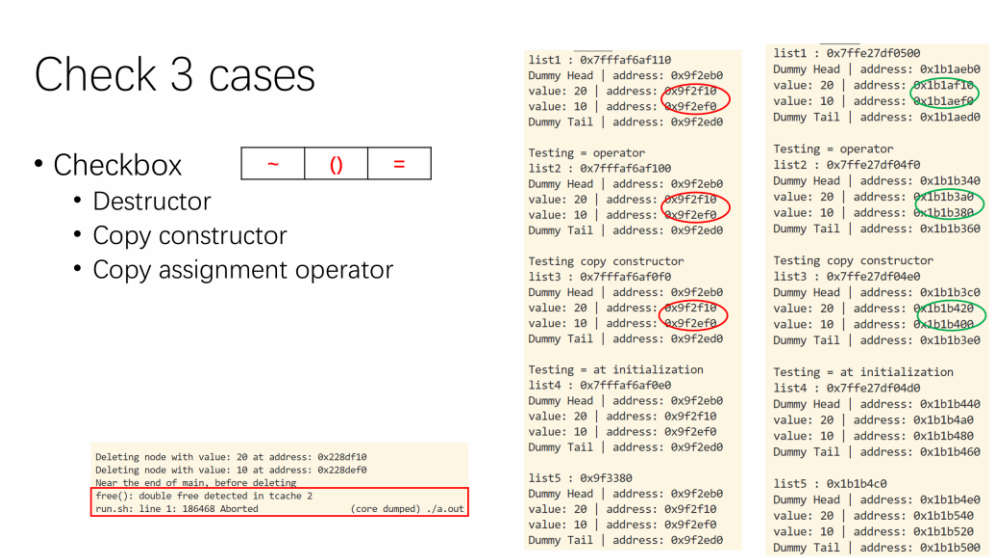
Lab4

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



Program2

stack

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

- Concrete/Regular Function
  - "CertainType1 FunctionName(CertainType2 parameter1, …)"
  - a fixed return type and fixed parameter types

```
int main(int argc, char const *argv[])
```

- 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>
T convertTo(U value)
{
    return static_cast<T>(value);
}
```

RPN(Postfix2Num)

- I/O hint
  - fstream

```
std::string input = argv[1];
std::ifstream file(input);
```

  - negative numbers
    - The first character is a '-'
    - Not a single '-' (It is the minus symbol)

```
77 -6 + 2 / 10 5 3
```

```
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?

```
8 5 - 4 * 3 - 3 / 2 * 4 * 16 * 512 -
```

Infix2Postfix

```
(((12 * (5 + 6)) - (8 * (5 - 3))) / ((4 - 2) * 2)) - (8 + 12)
```

- 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?
  - Parenthesis(right, ')')
    - Output & Pop until a left parenthesis is encountered

```
12 * 5 + 6 + 8 * 7
```

high low

```
(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( )
  i <- 0

  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]
      S.push( p[i]  )
    if p[i] == '('
      S.push( p[i] )
    if 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
```