This lecture explains homework 13 and homework 14. These two assignments use linked list for arithmetic operations.

Homework 13 evaluates postfix expressions.

Homework 14 converts infix expressions to postfix expressions.

We will use linked lists as stacks. This is a review of the insert function we explained earlier. Remember that this line p arrow next equal to h, this puts the newly created list node in front of the original list. This insert method creates a stack because the most recently added node is in front of the nodes inserted earlier.

We have also explained how to make a linked list a queue. Please review the fifth lecture about linked list.

What is infix expression? What is postfix expression?

In these two assignments, we will focus on binary operations. We will not consider unary operations. What is a unary operation? Negation is an example of a unary operation. It takes only one operand.

What is a binary operation? It takes two operands. For example, when we say 4 plus 9, 4 and 9 are operands. The plus sign is the operation for addition.

What is an infix expression? An infix expression puts the operation in the middle of the two operands.

In contrast, a postfix expression puts the operation after both operands. For humans, infix expressions are easier to read. For computers, postfix is easier. Postfix has three major advantages. First, postfix does not need parenthesis because the postfix expression already specifies the order of operation. Second, there is no need to worry about precedence. In arithmetic operations, multiplication and division have higher precedence than addition and subtraction. Third, evaluation of a postfix expression is easy to implement using a stack.

Let’s consider a few examples of postfix expressions. The first example is 3, 5, multiplication, 6, plus.

What does this mean? It is equivalent to 3 times 5 plus 6. The result is 15 plus 6 equals to 21.

The next example is 3 5 plus 6 multiplication. If we want to express this in the infix format, we need to use parentheses because multiplication has higher precedence than addition. The infix expression is open parenthesis, 3 plus 5, close parenthesis, multiplication 6. The result is 8 multiplied with 6, equals to 48.

The third example is 3, 5 , 6, plus, multiplication. This is another example that needs parentheses. The infix expression is 3 multiplication, open parenthesis, 5 plus 6, close parenthesis. The result is 3 multiplied with 11, equals to 33.

This lecture will talk about how to evaluate postfix expressions and then explains how to convert infix expressions to postfix expressions.

What is precedence in a C program? This table specifies the orders to execute operations. If several operators appear in the same statement, the precedence determines the order. In general, you do not need to memorize this table. You should make your programs very easy to write without checking this table. If you have multiple operations, break them apart into multiple statements in the correct order. If your programs are easy to understand, your programs are more likely to be correct.

Consider this example: 3, 4, 7, multiplication, plus, 2, minus. How should we evaluate this postfix expression?

To evaluate a postfix expression, when we see numbers, the numbers are pushed to a stack as shown here.

Number 3 is at the bottom of the stack. Number 4 is in the middle. Number 7 is at the top of the stack.

When an operator is encountered, the program pops the top two numbers, 4 and 7. We need to pop two numbers because this is a binary operation.

The operation is performed on the two numbers. This operator is multiplication. The two operands are 4 and 7. The result is 28. This number is pushed back to the top of the stack.

The stack now has two numbers: 28 at the top and 3 at the bottom.

Next, we encounter an operator again. We need to pop two numbers from the stack. They are 3 and 28. The operator is addition. The result is 31 and this number is pushed to the stack.

Number 2 is pushed to the top of the stack.

The next is an operator for subtraction. We need to pop two numbers from the stack. We need to be careful about the order because subtraction is not commutative.

What is commutative? It means whether we can change the order of the operands. For example, ei plus bee is the same as bee plus ei. .

Multiplication is also commutative: ei times bee is the same as bee times ei. .

Subtraction, however, is not commutative: ei minus bee is different from bee minus ei. .

Thus, we have to be careful about the order if the operator is subtraction.

We need to take the second number from the top of the stack as the first number for subtraction. The top number of the stack is the second number for subtraction. The result is 31 minus 2.

The answer 29 is pushed to the stack.

Nothing is left in the postfix expression. The final result is 29.

It is possible that the input has errors. For example, an operator must be after two operands. If your program sees and operator but the stack does not have two or more numbers, this is an error. One example is 7 plus. When the program sees the plus sign, there is only one number, 7, in the stack. This input is invalid as a postfix expression.

Another example is a single multiplication sign. The stack has no number. Thus, this is also invalid.

Another type of errors is too few operators. In this example, 2, 8, 3, plus. After reading the postfix expression, the stack has two numbers left: 11 at the top and 2 at the bottom. This is also an error because the stack should have only one number left.

Homework 14 asks you to convert infix to postfix. Let’s go over several example to understand how to convert.

Consider this example: 6 times 4 plus 3. We need to main a stack for the operator and a queue for the output. What is the difference between a stack and a queue? A stack is first-in, last out. A queue is first-in, first-out.

The first input of the expression is number 6. It is put to the output queue.

The next input is an operator. It is pushed to the operator stack.

The next input is a number. It is again put to the output queue.

The next input is an operator. It is addition. We need to compare this operator’s precedence with the one already in the operator stack.

Inside the stack, there is an operator already and it is multiplication. Multiplication has a higher precedence than addition.

Thus, the multiplication is popped from the stack and moved to the output. The addition is pushed to the operator stack.

This slide shows the operator stack and the output queue. Please notice that multiplication is at the output queue and addition is stored in the stack.

The next input is a number. It is stored in the output queue.

We have finished reading the input. There is still an operator in the stack. It should be popped from the stack and moved to the output queue.

This is the final result of the output queue. The postfix expression is 6, 4, multiplication, 3, addition.

The next example exchanges the locations of multiplication and addition. The input is 6, addition, 4, multiplication, 3. We know that multiplication should be done before addition. How can we express that in the postfix expression?

Again, we have an operator stack and an output queue.

The first input is a number. It is moved to the output queue.

The second is an operator. It is moved to the operator stack.

The third input is a number. It is moved to the output queue.

The next input is an operator for multiplication. We need to compare this operator with the addition operator already in the stack. Multiplication has higher precedence than addition. Thus, multiplication is pushed to the operator stack.

This slide reviews the earlier example. In the earlier example, we see multiplication first and then we see addition. For this case, multiplication is popped from the stack and addition is pushed to the stack.

Let’s go back to the example we were working on. At this moment, the output queue has two numbers: 6 and 4. The operator stack has two operators: multiplication at the top and addition at the bottom.

The next input is a number. It is moved to the output queue.

The input has nothing else left. We pop the operator stack and move multiplication to the output queue.

The last step pops the addition operator to the output queue.

The postfix expression is 6, 4, 3, multiplication, addition.

This slide compares the two examples we went through. If the input has 6, addition, 4, multiplication, 3. The postfix expression is 6, 4, 3, multiplication, addition.

If the input has 6, multiplication, 4, addition, 3. The postfix expression is 6, 4, multiplication, 3, addition.

The next example considers parentheses.

The input is six, multiplication, open parenthesis, 4, addition, close parenthesis.

The first input is a number. It is moved to the output queue.

The second is an operator for multiplication. It is moved to the operator stack.

The next input is the open parenthesis. It is moved to the operator stack.

The next input is a number. It is moved to the output queue.

The next input is an operator for addition. Because the top of the stack is an open parenthesis, the addition is pushed to the stack directly.

The next input is a number. It is put to the output queue.

The next input is a close parenthesis. We need to pop the operators in the stack until seeing the open parenthesis.

The addition operator is added to the output queue.

Since there is nothing left in the input, we pop the operator stack and add to the output queue.

The final result is 6, 4, 3, addition, multiplication.