**Data Structure**

**&**

**Algorithm**

**Class 8**

**Lab 11**

|  |
| --- |
| Lab Objectives:Simple Balanced ParenthesesBinary to decimal conversioninfix, prefix and postfix |

# Parentheses

## IMG_256A parenthesis is a tall, curvy punctuation mark used to set off material that isn't fundamental to the main topic

# Simple Balanced Parentheses

## We now turn our attention to using stacks to solve real computer science problems. You have no doubt written arithmetic expressions such as

## *(5+6)∗(7+8)/(4+3)*

## where parentheses are used to order the performance of operations.

## Balanced parentheses means that each opening symbol has a corresponding closing symbol and the pairs of parentheses are properly nested. Consider the following correctly balanced strings of parentheses:

(()()()())

(((())))

(()((())()))

## Compare those with the following, which are not balanced:

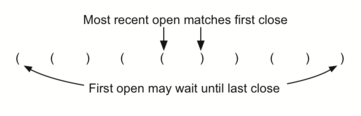
((((((())

()))

(()()(()

## The ability to differentiate between parentheses that are correctly balanced and those that are unbalanced is an important part of recognizing many programming language structures.

## This is a clue that stacks can be used to solve the problem.



# Balanced Symbols (A General Case)

## The balanced parentheses problem shown above is a specific case of a more general situation that arises in many programming languages. The general problem of balancing and nesting different kinds of opening and closing symbols occurs frequently. For example, in Python square brackets, [ and ], are used for lists; curly braces, { and }, are used for dictionaries; and parentheses, ( and ), are used for tuples and arithmetic expressions. It is possible to mix symbols as long as each maintains its own open and close relationship. Strings of symbols such as

{ { ( [ ] [ ] ) } ( ) }

[ [ { { ( ( ) ) } } ] ]

[ ] [ ] [ ] ( ) { }

## are properly balanced in that not only does each opening symbol have a corresponding closing symbol, but the types of symbols match as well.

## Compare those with the following strings that are not balanced:

( [ ) ]

( ( ( ) ] ) )

[ { ( ) ]

# Decimal Numbers

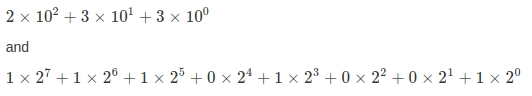
## IMG_256Decimal number system, also called Hindu-Arabic, or Arabic, number system, in mathematics, positional numeral system employing 10 as the base and requiring 10 different numerals, the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. It also requires a dot (decimal point) to represent decimal fractions.

# Binary Numbers

## IMG_256In mathematics and digital electronics, a binary number is a number expressed in the base-2 numeral system or binary numeral system, which uses only two symbols: typically 0 (zero) and 1 (one). The base-2 numeral system is a positional notation with a radix of 2. Each digit is referred to as a bit.

# Converting Decimal Numbers to Binary Numbers

## Integer values are common data items. They are used in computer programs and computation all the time. We learn about them in math class and of course represent them using the decimal number system, or base 10. The decimal number 23310 and its corresponding binary equivalent 111010012 are interpreted respectively as

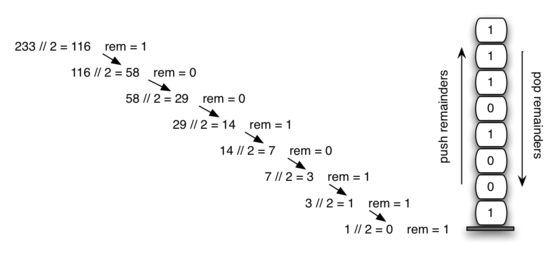


But how can we easily convert integer values into binary numbers? The answer is an algorithm called “Divide by 2” that uses a stack to keep track of the digits for the binary result.

## The Divide by 2 algorithm assumes that we start with an integer greater than 0. A simple iteration then continually divides the decimal number by 2 and keeps track of the remainder. The first division by 2 gives information as to whether the value is even or odd.

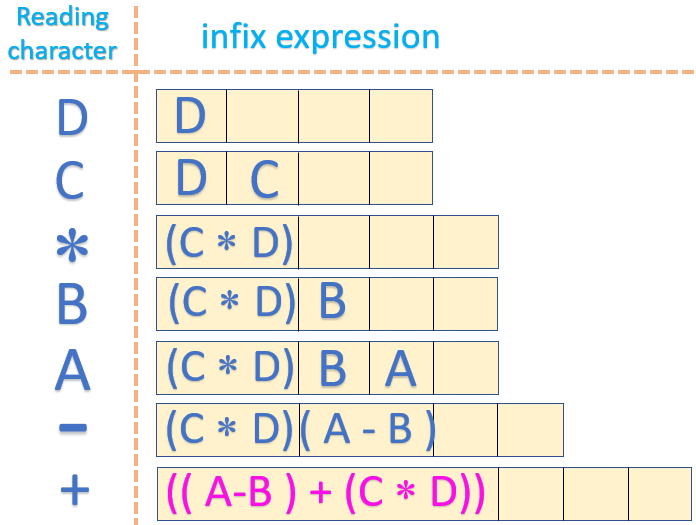
## An even value will have a remainder of 0. It will have the digit 0 in the ones place. An odd value will have a remainder of 1 and will have the digit 1 in the ones place. We think about building our binary number as a sequence of digits; the first remainder we compute will actually be the last digit in the sequence.

## As shown in below, we again see the reversal property that signals that a stack is likely to be the appropriate data structure for solving the problem.

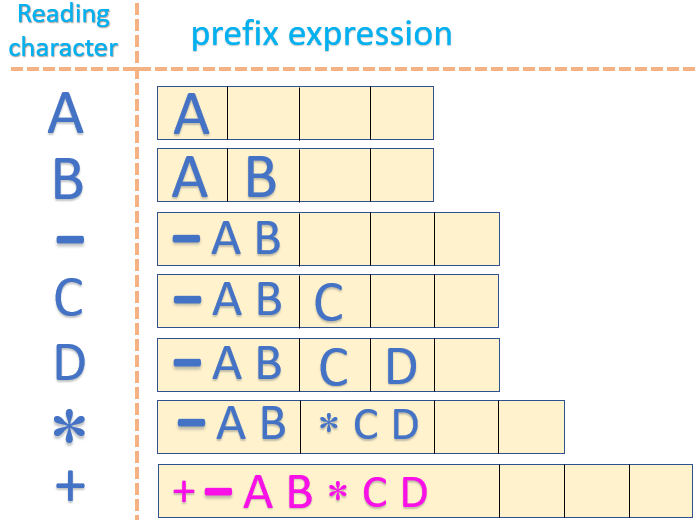


# Infix, Prefix and Postfix Expressions

## When you write an arithmetic expression such as B \* C, the form of the expression provides you with information so that you can interpret it correctly. In this case we know that the variable B is being multiplied by the variable C since the multiplication operator \* appears between them in the expression. This type of notation is referred to as infix since the operator is in between the two operands that it is working on.



## These changes to the position of the operator with respect to the operands create two new expression formats, prefix and postfix. Prefix expression notation requires that all operators precede the two operands that they work on



## IMG_256Postfix, on the other hand, requires that its operators come after the corresponding operands. A few more examples should help to make this a bit clearer

## Here are some examples of infix, prefix and postfix

|  |  |  |
| --- | --- | --- |
| ****Infix Expression**** | ****Prefix Expression**** | ****Postfix Expression**** |
| A+B | +AB | AB+ |
| A+(B\*C) | + A \* B C | A B C \* + |

Follow these Operator Precedence for calculating:

1. ( )
2. Multiplication / Division
3. Addition / Subtraction