

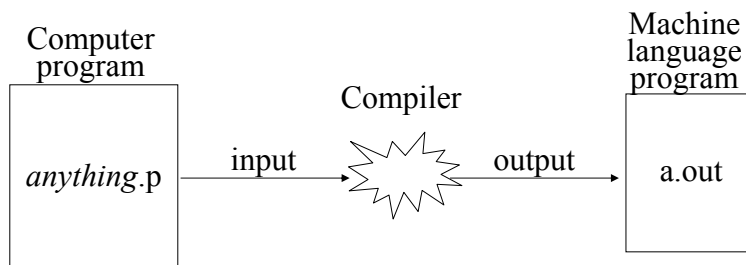
Theoretical Concepts For The Parsing Assignment

A return to the compilation process

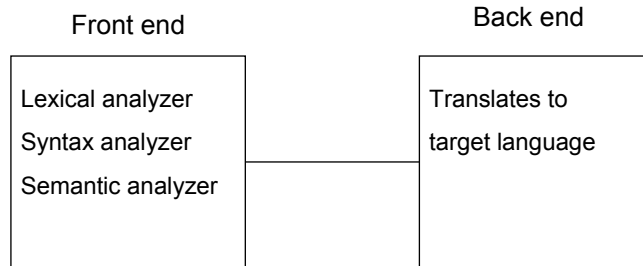
Parsing and Formal grammars

Divide and conquer through recursion

Compilation



Parts Of The Compiler



The Lexical Analyzer

Groups symbols together into entities

w h i l e

while

Syntax Analyzer (Parser)

Analyzes the structure of the program in order to group together related symbols

```
while (i < 5)
{
    statement;
    statement;
    :
}

while (i < 5)
    statement;
```

Semantic analyzer

Determines the meaning

```
int num;
double db;
```

Assignment 4

With a given English phrase your program must perform:

A lexical analysis

A syntactical analysis (parse the phrase)

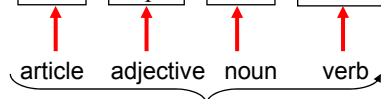
- Needs a formal grammar

A Perspective Into Assignment 4

User types in a sentence ₁

Perform a lexical analysis

Perform a syntactical analysis

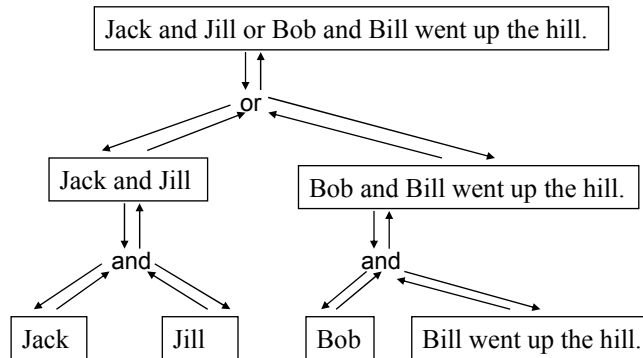


Is this sentence grammatically correct according to the rules of our syntax?

¹ The use of nicotine products by animals is not endorsed by this instructor or the University of Calgary

A Perspective Into Assignment 4 (2)

Conjunctions must be handled recursively



Backus-Naur Form (BNF)

An example of a formal grammar

Can be used in the fourth assignment to specify the syntax of the language (grammatically correct)

Introduced by Jim Backus and developed by Pete Naur to specify the syntax rules for Algol 60

Symbol	Meaning or usage
\diamond	To group related categories of information
	“OR” – to specify alternative options
::=	“As defined as” – a way of specifying a definition

BNF: General Examples

Example one:

$\langle A \rangle ::= \langle B_1 \rangle \langle B_2 \rangle \langle B_3 \rangle \dots \langle B_n \rangle$

Example two (alternatives):

$\langle A \rangle ::= \langle B_1 \rangle \langle B_2 \rangle \langle B_3 \rangle \mid \langle B_2 \rangle \langle B_4 \rangle \langle B_1 \rangle$

Example three (program specification):

$x = x + 1$

$\langle \text{Assignment statement} \rangle ::= \langle \text{variable} \rangle \langle = \rangle \langle \text{expression} \rangle$

BNF: Assignment 4

(The following specifications come from the main 233 course web page: www.cpsc.ucalgary.ca/~becker/233)

$\langle \text{STATEMENT} \rangle ::= \langle \text{Sentence} \rangle \langle \text{PUNCT} \rangle$

$\langle \text{Sentence} \rangle ::= \langle \text{NounPhrase} \rangle \langle \text{VerbPhrase} \rangle \mid$
 $\langle \text{NounPhrase} \rangle \langle \text{VerbPhrase} \rangle \langle \text{Conjunction} \rangle \langle \text{Sentence} \rangle$

$\langle \text{NounPhrase} \rangle ::= \langle \text{ProNoun} \rangle \mid \langle \text{ProperNoun} \rangle \mid \langle \text{Article} \rangle$
 $\langle \text{AdjectiveList} \rangle \langle \text{Noun} \rangle \mid \langle \text{Article} \rangle \langle \text{Noun} \rangle \mid \langle \text{Noun} \rangle$

BNF: Assignment 4




<VerbPhrase> ::= **<AdverbList>** <Verb> | <Verb> **<AdverbList>** |
 <Verb> **<NounPhrase>** **<AdverbList>** |
 <AdverbList> <Verb> **<NounPhrase>** |
 <Verb>

<AdjectiveList> ::= **<AdjectiveList>** <Adjective> | <nothing>

<AdverbList> ::= **<AdverbList>** <Conjunction> <Adverb> | <Adverb>

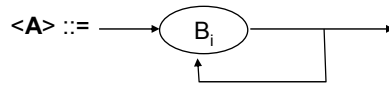
Syntax Diagrams

An alternative method for representing a formal language

Symbol	Meaning or usage
::=	“As defined as” – a way of specifying a definition
	A category of information that <i>can</i> be decomposed into its constituent subcategories
	A category of information that <i>cannot</i> be decomposed into constituent subcategories
	Indicates direction to read the flow of the diagram

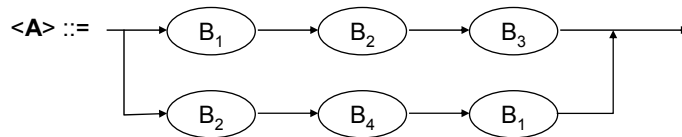
Syntax Diagrams: General Examples

Example one:



$i = 1, 2, 3, \dots, n$

Example two:



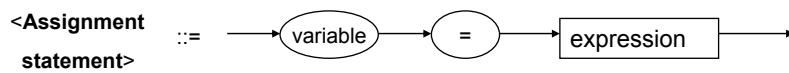
Parsing Assignment

James Tam

Syntax Diagrams: General Examples (2)

Example three (program specification)

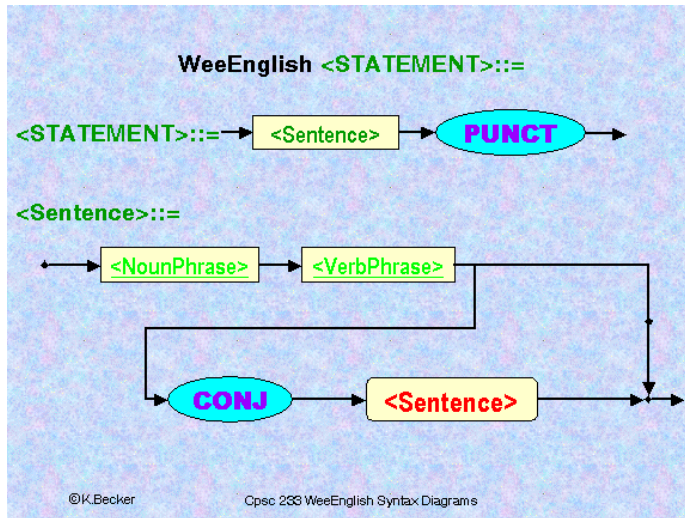
$x = x + 1$



Parsing Assignment

James Tam

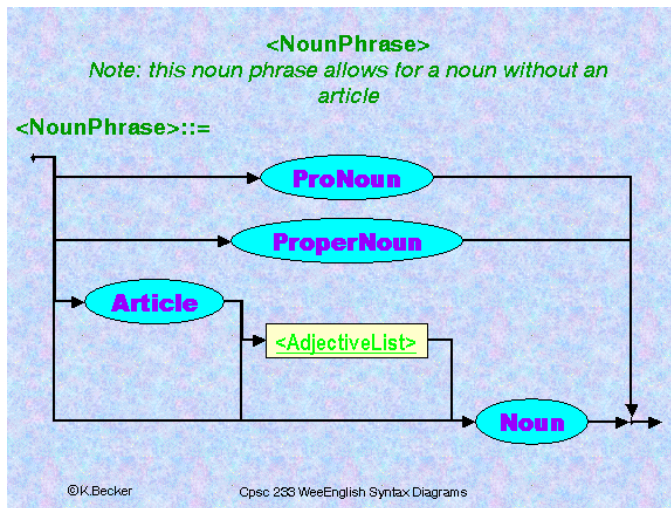
Syntax Diagrams: Assignment 4



Parsing Assignment

James Tam

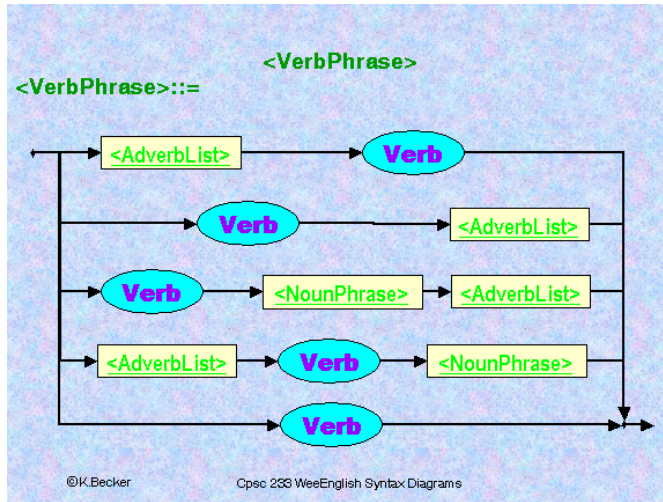
Syntax Diagrams: Assignment 4 (2)



Parsing Assignment

James Tam

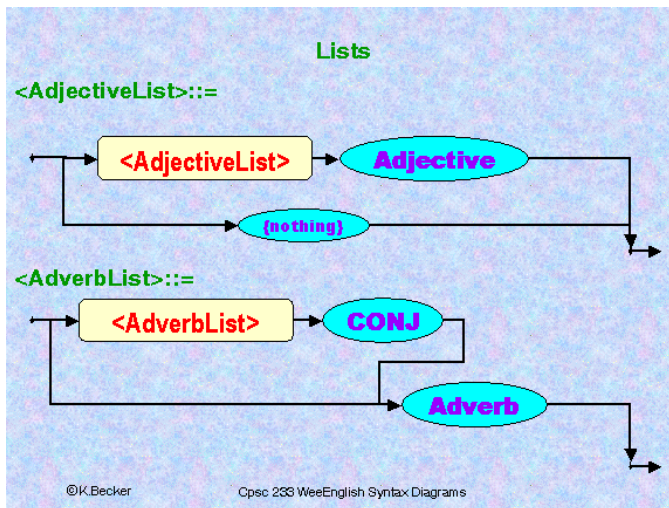
Syntax Diagrams: Assignment 4 (3)



Parsing Assignment

James Tam

Syntax Diagrams: Assignment 4 (4)



Parsing Assignment

James Tam

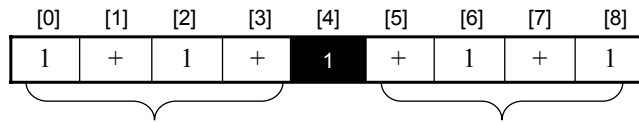
Divide And Conquer

- Split the problem into sub-problems (through recursive calls)
- Continue splitting each of the sub-problems into smaller parts until you cannot split the problem up any further
- Solve each of the sub-problems and combine the solutions yielding the solution to the original problem

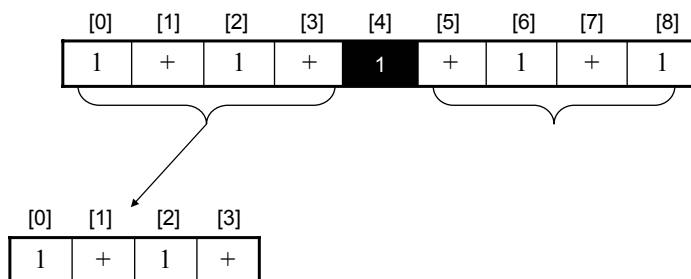
Divide And Conquer: An Example

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
1	+	1	+	1	+	1	+	1

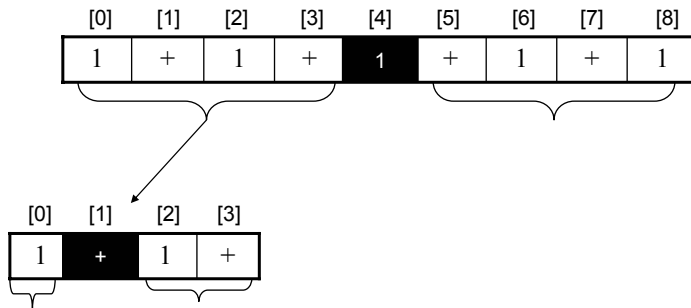
Divide And Conquer: An Example



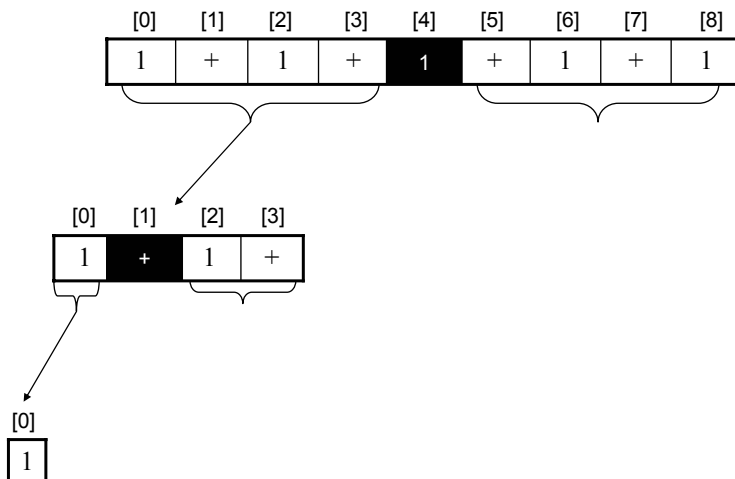
Divide And Conquer: An Example



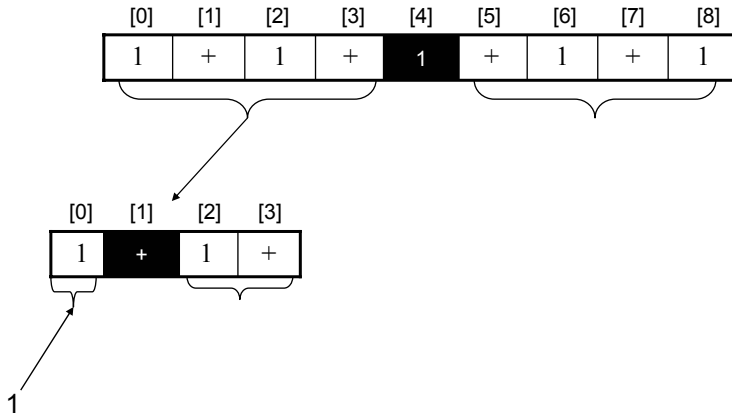
Divide And Conquer: An Example



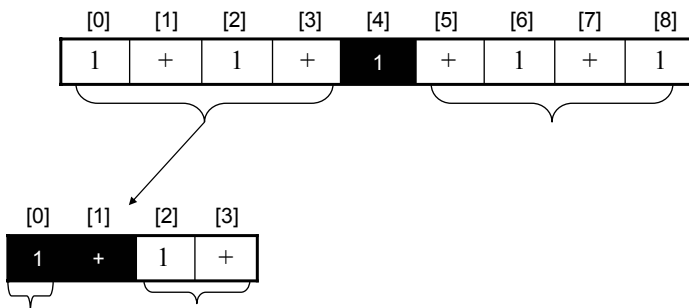
Divide And Conquer: An Example



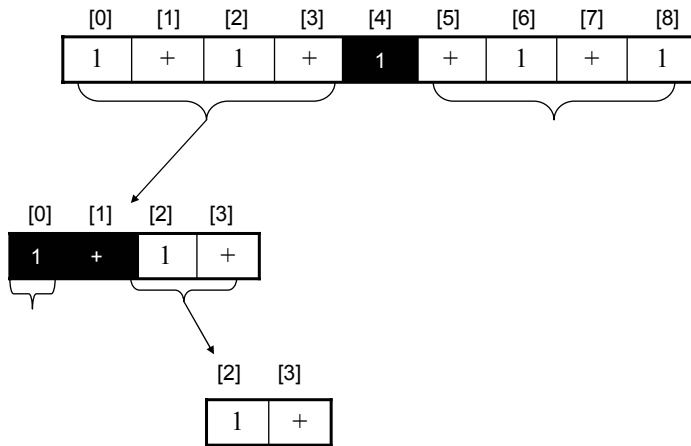
Divide And Conquer: An Example



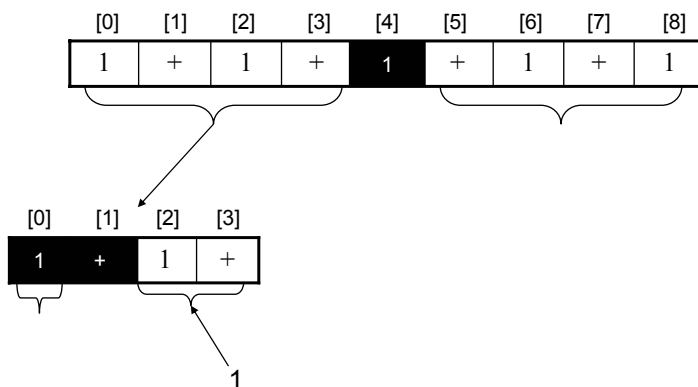
Divide And Conquer: An Example



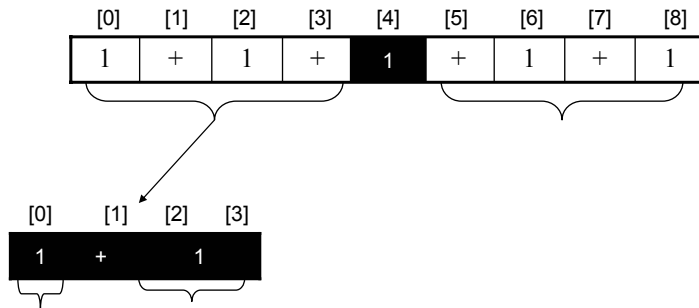
Divide And Conquer: An Example



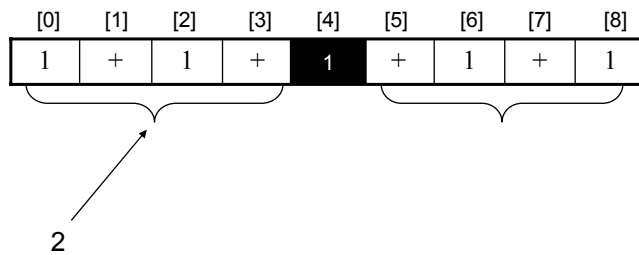
Divide And Conquer: An Example



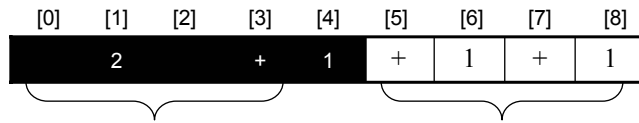
Divide And Conquer: An Example



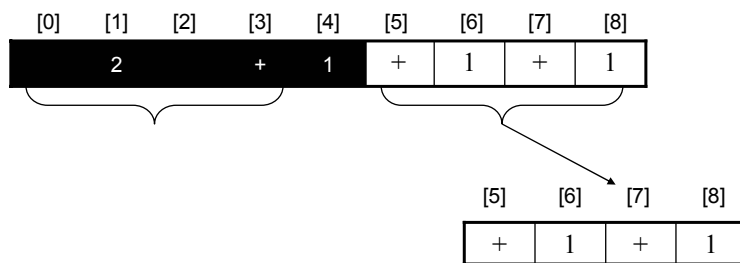
Divide And Conquer: An Example



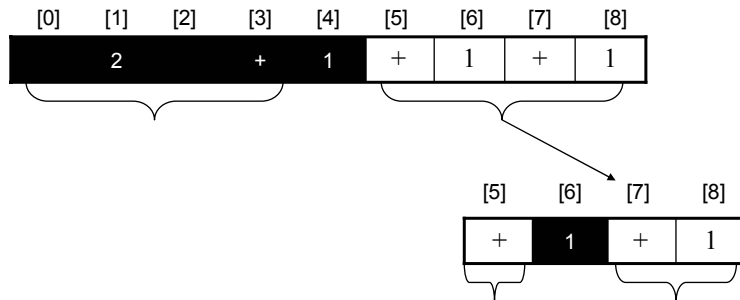
Divide And Conquer: An Example



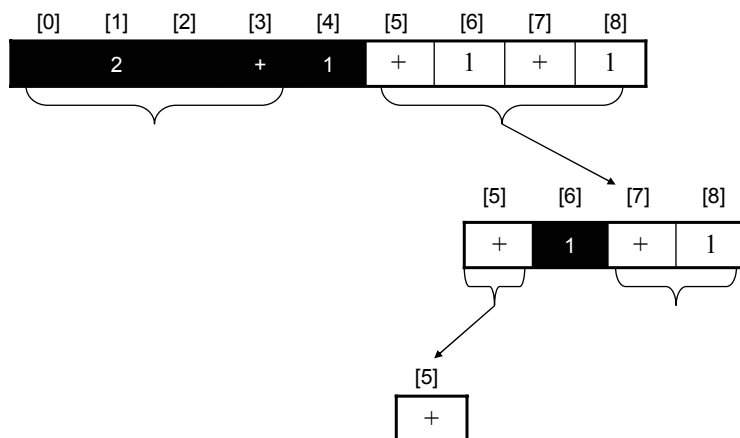
Divide And Conquer: An Example



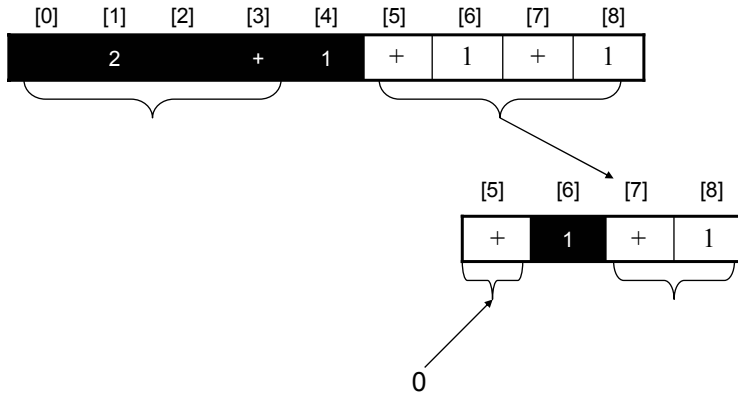
Divide And Conquer: An Example



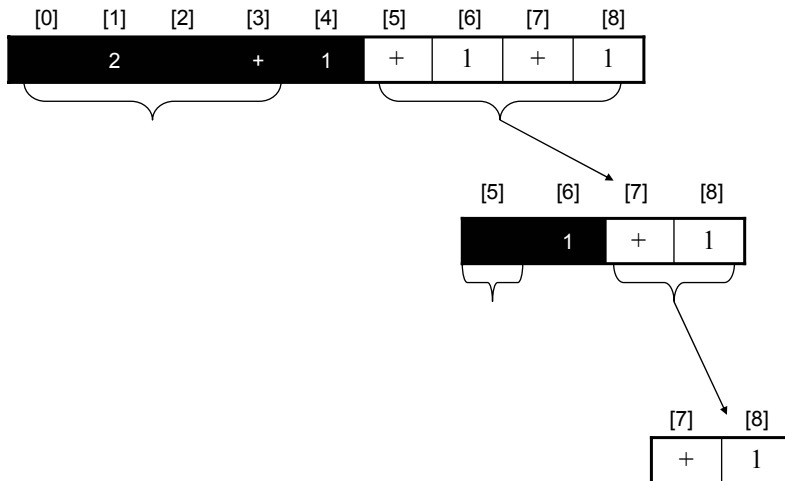
Divide And Conquer: An Example



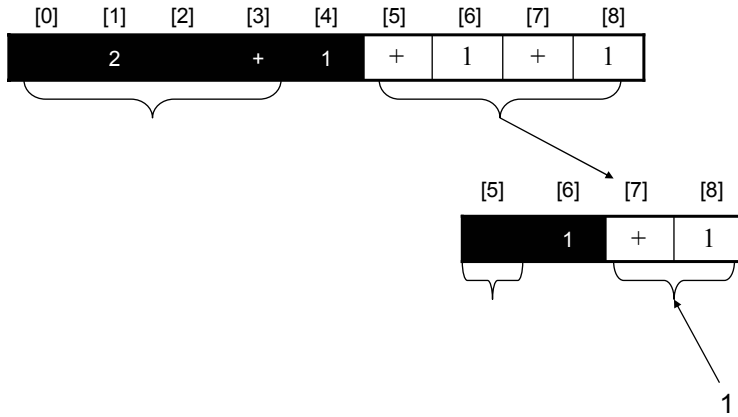
Divide And Conquer: An Example



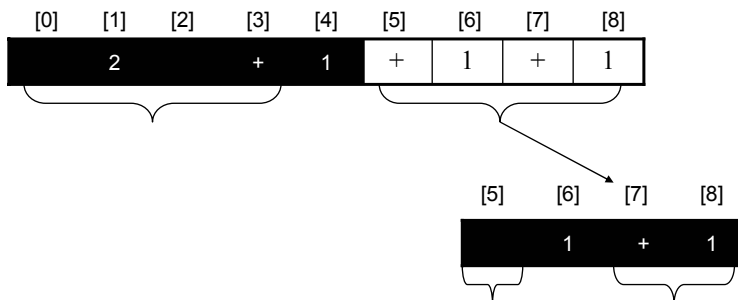
Divide And Conquer: An Example



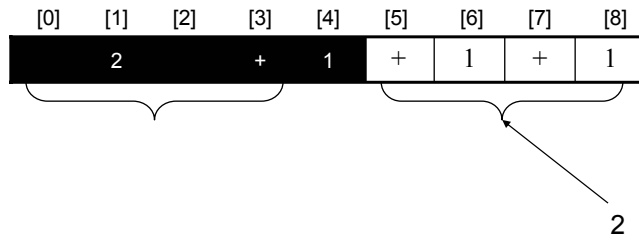
Divide And Conquer: An Example



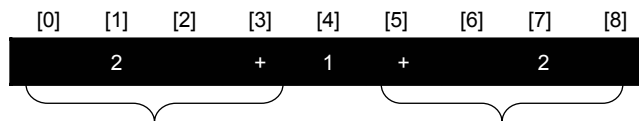
Divide And Conquer: An Example



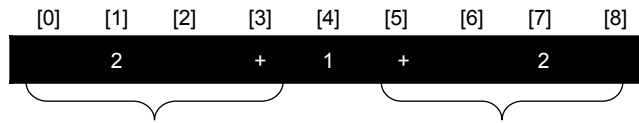
Divide And Conquer: An Example



Divide And Conquer: An Example



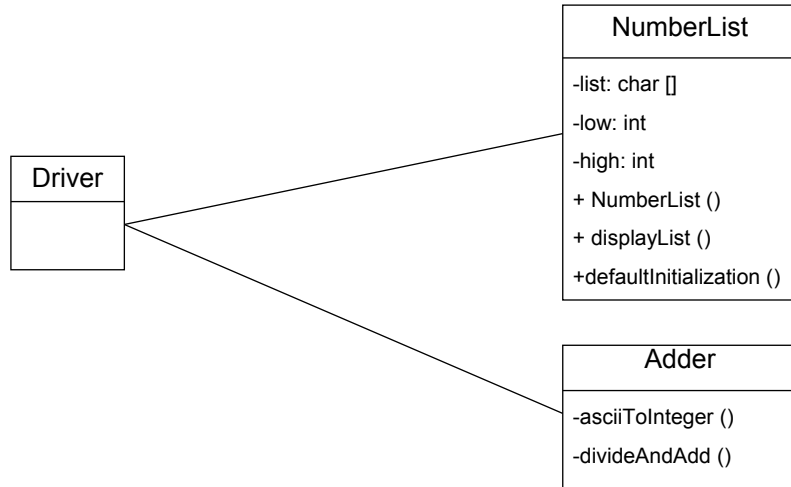
Divide And Conquer: An Example



Divide And Conquer: An Example

Final result after recursive calls = 5

Divide And Conquer Example



Divide And Conquer Example

The complete source and executable files can be found in
Unix under the directory:

`/home/profs/tamj/233/examples/recursion`

The Driver Class

```
class Driver
{
    public static void main (String [] argv)
    {
        NumberList list = new NumberList ();
        Adder listAdder = new Adder();
        int total = listAdder.divideAndAdd(list.getList(),
                                         list.getLow(),
                                         list.getHigh());

        System.out.println();
        System.out.println("SUM OF LIST..." + total);
        System.out.println();
    }
}
```

The NumberList Class

```
class NumberList
{
    private char [] list;
    private int low;
    private int high;

    public NumberList ()
    {
        int i, noElements;
        System.out.print("Enter number of array elements: ");
        high = Console.in.readInt();
        Console.in.readChar();
        high = high - 1;
        low = 0;
        list = new char [high+1];
        defaultInitialization();
        displayList();
    }
}
```


The NumberList Class (2)

```
public void defaultInitialization ()
{
    int i;
    for (i = low; i <= high; i++)
    {
        if (i % 2 == 0)
            list[i] = '1';
        else
            list[i] = '+';
    }
}
```

The NumberList Class (3)

```
public void displayList ()
{
    int i;
    System.out.println();
    System.out.print("Displaying the number list: ");
    System.out.println (list);
    System.out.println();
}
```

Class Adder: AsciiToInteger

```
class Adder
{
    private int asciiToInteger (char ch)
    {
        int temp;
        // Recall that the ascii value for the character '0' is 48.
        temp = (int) (ch - 48);
        return temp;
    }
}
```

Class Adder: DivideAndAdd

```
class Adder
{
    public int divideAndAdd (char [] array, int low, int high)
    {
        System.out.println("SUBDIVIDED ARRAY: " + "low=" + low + " " +
            "high=" + high);
    }
}
```

Class Adder: DivideAndAdd (2)

```
// THREE BASE CASES:  
// One element in sublist: convert from char to int and return if it's a number.  
if (low == high)  
{  
    if (array[low] != '+')  
    {  
        int temp = asciiToInteger(array[low]);  
        return temp;  
    }  
    else  
    {  
        // It's a plus sign don't sum the ascii value.  
        return(0);  
    }  
}
```

Class Adder: DivideAndAdd (3)

```
// Two elements in sublist  
if ((low+1)==high)  
{  
    // Order of elements: operation, operand  
    if (array[low] == '+')  
    {  
        int temp = asciiToInteger (array[high]);  
        return temp;  
    }  
    // Order of elements: operand, operation  
    else if (array[high] == '+')  
    {  
        int temp = asciiToInteger (array[low]);  
        return temp;  
    }  
}
```

Class Adder: DivideAndAdd (4)

```
// Three elements in sublist
if ((low+2) == high)
{
    // Order of elements: <operand> <operation> <operand>
    if (array[low] != '+')
    {
        int operand1 = asciiToInteger(array[low]);
        int operand2 = asciiToInteger(array[high]);
        return (operand1+operand2);
    }
    // Order of elements: <operation> <operand> <operation>
    else
    {
        int temp = asciiToInteger(array[low+1]);
        return temp;
    }
}
```

Class Adder: DivideAndAdd (5)

```
// RECURSIVE CASES:

// More than four elements in the list.
int middle, leftTotal, rightTotal, total;
int leftLow, leftHigh, rightLow, rightHigh;
middle = (int) ((low+high)/2);

// Set low and high bound for the left sublist.
leftLow = low;
leftHigh = middle - 1;

// Set low and high bound for the right sublist.
rightLow = middle + 1;
rightHigh = high;

leftTotal = divideAndAdd(array, leftLow, leftHigh);
rightTotal = divideAndAdd(array, rightLow, rightHigh);
```

Class Adder: DivideAndAdd (7)

```
total = leftTotal + rightTotal;

if (array[middle] != '+')
    total = total + asciiToInteger(array[middle]);

// Recursive calls finished.
return total;
}
}
```

Summary

You should now know:

- Compilation: What are the major parts of a compiler
- How formal grammars can be used to specify the syntax of a language
- Two examples of specifying syntax rules
 - Backus-Naur form (BNF)
 - Syntax diagrams
- Divide and Conquer through recursion