**Programs are strings**

Consider the following C example

void main(void) {

char \*message [] = {“Hello “, “World”};

int I;

for(i=0; I < 2; ++i)

printf(“%s”, message[i]);

printf(“\n”);

}

Code might as well appear as

void main(void) { char \*message[] = {“Hello “, “World”}; int I; for(I = 0; I < 2; ++i) printf(“%s”, message[i]; printf(“\n”);

Lisp Example

(defun hello

(print

(cons `Hello (list `World))))

Compiler sees (defun hello (print (cons `Hello (list `World))))

**Formal Grammars**

* Grammars generate languages
* Syntax of programming language may be described through specification pf a grammar
* Many types of grammars (later)

**An Alphabet and Forming Strings**

An alphabet (V) is a finite, nonempty set of *symbols*

*V* = {*a, b, c, … , z*}

The concatenation of *a* and *b*, denoted *a* o *b*, produces a sequence of two symbols simply denoted hereafter as *ab*.

A *string* over *V* is either a single symbol from *V* or a sequence of symbols formed by concatenation of zero or more symbols from *V*. Therefore, from *V* above, ‘*a’*, ‘*ab’,* ‘*az*’, and ‘*azab’,* are strings or *sentences*, over v

A string has a natural ordering of elements from left to right

Often it is convenient to denote a string like *x = aaa…a*, where a symbol (or sequence of symbols) is requested *n* times as *x = an.* For example:

*aabbbcccc* = *a*2*b*3*c*4

Define *V*+ as

V+ = *V U V2 U V3U ….*

*V+ is the set of all nonempty sentences producible using V. Adding the smpty string to V+produces V\*, ie*

*V\* = {ϵ} …*

**Languages using strings**

are used with V to give meaning to a subset of strings, L c\_ V\*

L is called a language. Furthermore

* Languages are generated by grammars.
* Another viewpoint is that a grammar restricts the production of strings from V.
* Grammars are used to recognize (parse) elements of a language

Grammar: Formal Definition

A *grammar*

G = (VT, VN, P, S)

consists of the following four entities:

1. A set of terminal or primitive symbols (primitives), denoted VT.

Grammar Application Modes

Grammar may be used in two modes:

1. Generative: Used to create a string of terminal symbols using P; a sentence in the language of the grammar is thus generated. Programming.
2. Analytic: Given a sentence, together with specification of G, one seeks to determine
   1. If sentence was generated by G, and if so,
   2. The structure (usually characterized as the sequence of

Languages, Possible Strings, and L(G)

* Any subset L C\_ VT\* is a language
* If |L| is finite, the language is called finite, otherwise it is infinite.

Memorize:

* The language generated by grammar G, denoted L(G) I the set of all strings which satisfy

1. Each string consists solely of terminal symbols from VT of G; and
2. Each string was poruced from S using P of G.

int main (void) { // Satisfies 1 and 2

int x;

main int (void) { // Only satisfies 1

int x;

**A sample *GS:* Type and L(GS)**

S → AB In BNF:

S → C <s> ∷= <a><b> | <c>

A → C <a> ∷= <c> | a

A → a <b> ∷= b | c

B → b <c> ∷= d

B → c

C → d → means “can be replaced by”

S is the starting symbol and VT and VN may be deduced from the

S S S S S

AB AB C AB AB

CB Ab d aB ACB

Cb Cb ac ACCB

db db addb

L(G) { db L(G) { db L(G) { d L(G) { ac

1. Symbols beginning with capital letter are Vn
2. Lowercase letter are VT
3. n denotes length of string s, ie
   1. n = |s|
4. Greek letters represent (possibly empty) strings, typically comprised of terminals and/or nonterminals

T1: Context-sensitive

A T1 or context sensitive grammar restricts productions to

ααiβ → αβ*i*β

meaning βi replaces αi­ in the context of α and β, where α, β E- (VN U TT)\*, αi E- VN and βi E- (VN U VT)\* - {ϵ}. Note that α or β (or both) may equal ϵ.

Furthermore, |ααiβ| <\_ |αβiβ|. Replacements involving empty string treated as special case.