Assignment

Basic attribute for imperative languages, although syntax differs:

A := B  Pascal, Ada
A = B    C, FORTRAN, PL/I, Prolog, ML, SNOBOL4
MOVE B TO A COBOL
A ← B   APL
(SETQ A B) LISP

Note that LISP, an applicative language, has an assignment. Most LISP programmers do not write purely applicative programs.

C has multiple ways to assign: A=B, A++, ...
• How many C assignment operations can you identify?
Explicit sequence controls

Early languages used labels on statements and explicitly determined location of branches (early FORTRAN):

```
IF (A.GT.7) GOTO 100
...
100 CONTINUE ↔ Explicit control
```

This is the “goto” controversy of the early 1970s. Languages do not need explicit control and concept is not used or needed even if present.

• C includes a break statement for exit out of a loop. It is a limited form of goto, that also has its problems with correctness.
Control structures

Basic control structures: Programs need 4 basic structures:

• A sequencing between statements
• A conditional for choosing alternatives
• A Loop construct
• A function call for calling subroutines. (We consider subprograms elsewhere in this course.)
Sequencing

Compound statement: Typical syntax:

begin
statement1;
statement2;
...
end;

Execute each statement in sequence.

Sometimes (e.g., C) { ... } used instead of
begin ... end
Conditional

The if statement is typical:

if expression then statement1 else statement2

if expression then statement1

The expression is evaluated and if true, the then clause is executed, otherwise the else clause is executed. If there is no else clause, then the next statement is executed.

• Since a begin statement is a statement, the then or else clause can be an arbitrary number of commands.

As with everything else in C, its austere syntax for if is: if (expression) statement1 else statement2;
Iteration

Three iterations are most common:

1a. while expression do statement; - Evaluate expression and if true execute statement. Then repeat process.

1b. Repeat statement until expression; - Execute statement and then evaluate expression. Quit if expression is now true.

1c. Indefinite iterations (Ada):

   loop
       exit when condition
   end loop;

2. For loop - Specify a count of the number of times to execute a loop:
   - for I=1 to 10 do statement;
   - for(I=0;I<10; I++) statement;
   - perform statement 10 times;

3. Non-ordinal set
   - foreach $X(@arrayitem){statement} - Perl
Case statement

A form of multiway branch (similar to if):

```plaintext
case Tag is
    when 0 => begin
        statement0
    end;
    when 1 => begin
        statement1
    end;
    when 2 => begin
        statement2
    end;
    when others => begin
        statement3
    end;
end case
```
Implementation of case

Evaluate CASE branch

Jump table

Alternatives for CASE statement

Instructions for statements preceding CASE statement

Fetch value of variable TAG

Jump to L0+t

L0:

Jump to L1

Jump to L2

Jump to L3

Jump to L4

Jump to L4

Jump to L4

L1:

Instructions for statement 0

Jump to L5

L2:

Instructions for statement 1

Jump to L5

L3:

Instructions for statement 2

Jump to L5

L4:

Instructions for statement 3

L5:

Instructions for statements following CASE statement
Understanding control structures

Earlier discussion on control structures seemed somewhat ad hoc.

Is there a theory to describe control structures?

Do we have the right control structures?

Roy Maddux in 1975 developed the concept of a prime program as a mechanism for answering these questions.
Spaghetti code

(a) Spaghetti code

(b) Structured code
Control structures represented as flowcharts

SEQUENCE
S1; S2; S3

CONDITIONAL
If–Then  If P then S;
if–then–else: if p then S1
else S2

CASE
case p of
L1: S1;
L2: S2;
L3: S3;
end
ITERATION

while P do S

repeat S until p

Note following problem: Read data from a file and process until end of file.

Do <while more to process>
  process it
  end

Usual implementations:
1. while <not reached end of data> do
   Read data
   if not <end of file> then
     Process data
   Read data
   Requires 2 tests for end of file

2. Read data
   while not <end of file> do
     Process data
     Read data
   Requires 2 reads

DO WHILE DO

do S1 while P do S2

do Read data
   while more to read
   process data
   end
Theory of prime programs

Consider 3 classes of flowchart nodes:

- Function node
- Predicate node
- Join node

Any flowchart is a graph of directed arcs and these 3 types of nodes:
Proper programs

A proper program is a flowchart with:

- 1 entry arc
- 1 exit arc
- There is a path from entry arc to any node to exit arc

A prime program is a proper program which has no embedded proper subprogram of greater than 1 node. (i.e., cannot cut 2 arcs to extract a prime subprogram within it).

A composite program is a proper program that is not prime.
Prime flowcharts

(a) Prime Program

(b) Composite Program
Prime decomposition

Every proper program can be decomposed into a hierarchical set of prime subprograms. This decomposition is unique (except for special case of linear sequences of function nodes).
All prime programs can be enumerated

All primes can be enumerated (next slide)

Each implements a function.

What is function for 2-node primes (c) and (d) and 4-node primes (l) through (q)?

All primes with at least 1 function node are our usual control structures - except (k) - the do-while-do construct. More about this later.
Number of primes is infinite

Construction to create primes with any number of nodes:

Use of prime programs to define structured programming:
Concept first used by Dijkstra in 1968 as gotoless programming.

Called structured programming in early 1970s—
Program only with if, while and sequence control structures.
Structured programming

Issue in 1970s: Does this limit what programs can be written?
• Resolved by Structure Theorem of Böhm-Jacobini.

Here is a graph version of theorem originally developed by Harlan Mills:

![THEOREM: Only if, while, and sequence control structures are necessary to create a flowgraph functionally equivalent to any other flowgraph.](image)
Structure theorem

Outline of proof:
1. Label each arc with 1 being the entry arc and 0 the exit arc.
2. Let I be a new variable not used in program.
3. For each node in program construct the following:

(a) For:

(b) For:

(c) For:

<table>
<thead>
<tr>
<th>i</th>
<th>f</th>
<th>j</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>p</th>
<th>j</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>k</th>
<th>j</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>l := j</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>p</th>
<th>l := j</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>p</th>
<th>l := k</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>l := k</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>l := k</th>
</tr>
</thead>
</table>
Structure theorem (continued)

4. Create the following new flowchart on right.

If F is the original function, what is the new function?
Program verification - again

For every flowchart, we can compute a function for each arc in its prime decomposition.

\[ f_1: S_1 \to S_3; \]
\[ f_2: S_2 \to S_4; \]
\[ f_3: S_5 \to S_f \]

A program is then \( C(f_1, f_2, f_3): S_0 \to S_f \)

Program verification is a formal method to define these functions:

- Axiomatic – Hoare, 1969
- Weakest precondition – Dijkstra, 1972
- Algebraic data types – Guttag, 1975
- Functional correctness – Mills, 1975
Prime programs - Summary

Structure theorem does not say:
Given any “spaghetti program” convert it into a structured program.

What it does say is that there is no loss of functionality in using only those control structures. It is still up to the programmer to choose the best algorithm to solve problem.

Also, the prime decomposition shows that the primes up to 4 nodes are the useful programming language control structures, except for the do-while-do, which has been ignored by language designers.