Lecture 8: Control Structures I
Intro. to Programming, lecture 8: Control structures I
The notion of algorithm

General definition:

An **algorithm** is the specification of a process to be carried out by a computer.
Not quite an algorithm

Preparazione e tempi di cottura
Zubereitung - Preparation

Versate le verdure ancora surgelate in 1 litro abbondante d'acqua fredda con 2 cucchiai d'olio, salate e cuocete secondo i tempi indicati.

Tiefgekühlte Gemüse in einen Liter kaltes Wasser geben, 2 Esslöffel Öl und Salz hinzufügen.

Verser les légumes surgelés dans 1 litre d'eau froide, ajouter deux cuillers à soupe d'huile et du sel.

45 min.

15 min.
5 properties of an algorithm

Defines data to which process will be applied

Every elementary step taken from a set of well-specified actions

Describes ordering(s) of execution of these steps

2 and 3 based on precisely defined conventions, suitable for execution by an automatic computer

For any data, guaranteed to terminate after finite number of steps
Algorithm vs program

“Algorithm” usually considered a more abstract notion, independent of platform, programming language etc.

In practice, the distinction tends to fade:
- Algorithms need a precise notation
- Programming languages becoming more abstract

However:
- In programs, data (objects) are just as important as algorithms
- A program typically contains many algorithms and object structures
What makes up an algorithm

Basic steps:

- Feature call \( x.f(a) \)
- Assignment
- ...

Sequencing of these basic steps:

CONTROL STRUCTURES
Control structures

Definition: program construct that describes the scheduling of basic actions

Three fundamental control structures:

- **Sequence**
- **Loop**
- **Conditional**

They are the “Control structures of Structured Programming”
Sequence: “To achieve $C$ from $A$, first achieve an intermediate goal $B$ from $A$, then achieve $C$ from $B$”

Loop: solve the problem on successive approximations of its input set

Conditional: solve the problem separately on two or more subsets of its input set
The sequence (or Compound)

\[\text{instruction}_1\]

\[\text{instruction}_2\]

\[\ldots\]

\[\text{instruction}_n\]
Semicolon as optional separator

\[ \text{instruction}_1; \]

\[ \text{instruction}_2; \]

\[ \ldots ; \]

\[ \text{instruction}_n \]
Not quite an algorithm

PREPARAZIONE E TEMPI DI COTTURA
ZUBEREITUNG - PREPARATION

Versate le verdure ancora surgelate in 1 litro abbondante d'acqua fredda con 2 cucchiaini d'olio, salate e cuocete secondo i tempi indicati.

Tiefgegorene Gemüse in einen Liter kaltes Wasser geben, 2 Esslöfvel Öl und Salz hinzufügen.

Verser les légumes surgelés dans 1 litre d'eau froide, ajouter deux cuillers à soupe d'huile et du sel.

45 min.

15 min.
Correctness of a **Compound**

Precondition of $instruction_1$ must hold initially

Postcondition of each $instruction_i$ must imply precondition of each $instruction_{i+1}$

Final effect is postcondition of $instruction_n$
Conditional instruction

if

  Condition

then

  Instructions

else

  Other_instructions

end
Computing the greater of two numbers

if $a > b$
then
  $max := a$
else
  $max := b$
end
As a function

\[
\text{maximum}(a, b : \text{INTEGER}): \text{INTEGER}
\]

-- The higher of \(a\) and \(b\)

do

if
    \(a > b\)
then
    \text{Result} := a
else
    \text{Result} := b
end

done
The conditional as problem-solving technique

PROBLEM SPACE

Region 2

Use technique 2

Region 1

Use technique 1
Basic form

if \textit{Condition} then

\textit{Instructions}

else

\textit{Other\_instructions}

end
A variant of the conditional

\[
\text{if } \textit{Condition} \text{ then} \\
\textit{Instructions} \\
\text{end}
\]
A variant of the conditional

```
if Condition then
  Instructions
end
```

(Means the same as

```
if Condition then
  Instructions
else
  end
end
```

with an empty “else” clause)
Nesting

if \text{Condition}_1 \text{ then }
\text{Instructions}_1
else
  if \text{Condition}_2 \text{ then }
    \text{Instructions}_2
  else
    if \text{Condition}_3 \text{ then }
      \text{Instructions}_3
    else
      \ldots
    end
end
end
Nested structure
Comb-like structure
Comb-like conditional

\[
\text{if } \text{Condition}_1 \text{ then } \\
\quad \text{Instructions}_1 \\
\text{elseif } \text{Condition}_2 \text{ then } \\
\quad \text{Instructions}_2 \\
\text{elseif } \text{Condition}_3 \text{ then } \\
\quad \text{Instructions}_3 \\
\text{elseif} \\
\quad \ldots \\
\text{else} \\
\quad \text{Instructions}_0 \\
\text{end}
\]
Comb-like structure

```
if condition_1 then
  Instructions_1
elseif condition_1 then
  Instructions_2
  ...
else
  Instructions_0
end
```
On the menu for today

- Loops and their invariants

- See what it takes to ensure that a loop terminates

- Look at the general problem of loop termination

- Examine lower-level control structures: “Goto” and flowcharts; see rationale for the “control structures of Structured Programming”

- Prove the undecidability of the Halting Problem
Loop

from
  Initialization
  -- Compound
until
  Exit_condition
  -- Boolean_expression
loop
  Body
  -- Compound
end
Loop, full form

\[
\text{from} \quad \text{Initialization} \quad \text{-- Compound}
\]

\[
\text{invariant} \quad \text{Invariant_expression} \quad \text{-- Boolean_expression}
\]

\[
\text{variant} \quad \text{Variant_expression} \quad \text{-- Integer_expression}
\]

\[
\text{until} \quad \text{Exit_condition} \quad \text{-- Boolean_expression}
\]

\[
\text{loop} \quad \text{Body} \quad \text{-- Compound}
\]

\[
\text{end}
\]
Another loop syntax

```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.\n");
    return 0;
}
```
Loop, full form

from
  Initialization -- Compound

invariant
  Invariant_expression -- Boolean_expression

variant
  Variant_expression -- Integer_expression

until
  Exit_condition -- Boolean_expression

loop
  Body -- Compound

end
Looping over stations of a line

```
from fancy.start
until fancy.after
loop
   -- "Do something with fancy.item"
   fancy.forth
end
```

Previously and in the textbook: `fancy_line`
Operations on a list

before

item

index

count

after

1

start

back

forth

(The cursor)

Commands

Queries

(boolean)
Operations on a list

before

1

start

item

index

forth

(The cursor)

count

after

Intro. to Programming, lecture 8: Control structures I
Looping over stations of a line

```
from
    fancy.start
until
    fancy.after
loop
    -- “Do something with fancy.item”

    fancy.forth
end
```
Displaying station names

from
  fancy.start
until
  fancy.after
loop
  -- Display name of next station:
  Console.show(fancy.item)
  fancy.forth
end
Computing the “maximum” of station names

```pascal
from
  fancy.start; Result := ""
until
  fancy.after
loop
  Result := greater (Result, fancy.item.name)

  fancy.forth
end
```
Assignment

Result := "XYZ"

-- Change the value of Result to "XYZ"
Computing the “maximum” of station names

from
  fancy.start; Result := ""
until
  fancy.after
loop
  Result := greater (Result, fancy.item.name)
  fancy.forth
end

Greater

The greater of two strings, alphabetically, e.g.

greater ("ABC", "AD") = "AD"
In a function

\[
\text{highest\_name: STRING is}
\]

-- Alphabetically greatest station name of line

\[
do
\]

\[
\text{from}
\]

\[
fancy\_start; \text{Result} := ""
\]

\[
\text{until}
\]

\[
fancy\_after
\]

\[
\text{loop}
\]

\[
\text{Result} := \text{greater} (\text{Result}, fancy\_item\_name)
\]

\[
fancy\_forth
\]

\[
\end
\]

\[
\end
\]
highest_name: STRING is
   -- Alphabetically greatest station name of line

   do
      from fancy.start; Result := ""
      until fancy.after
      loop
         Result := greater(Result, fancy.item.name)
      end loop
   end do

   ensure
      Result /= Void and then not Result.empty
   end ensure

Intro. to Programming, lecture 8: Control structures I  40
Loop as approximation strategy

Result = \(name_1\)

Result = Max (\(name_1, name_2\))

Result = Max (\(name_1, name_2, ..., name_i\))

Result = Max (\(name_1, name_2, ..., name_i, ..., name_n\))
Computing the “maximum” of station names

from
    fancy.start; Result := ""

until
    fancy.after
loop
    Result := greater(Result, fancy.item.name)
    fancy.forth
end
The loop invariant

\[
\text{from} \quad \text{fancy.start} ; \text{Result} := ""
\]

\text{invariant}

\[
\begin{align*}
\text{fancy.index} & \geq 1 \\
\text{fancy.index} & \leq \text{fancy.count} + 1
\end{align*}
\]

\text{-- Result is the alphabetically highest of the names of previous stations}

\text{until} \quad \text{fancy.after}

\text{loop}

\text{Result} := \text{greater} (\text{Result}, \text{fancy.item.name})

\text{fancy.forth}

\text{end}
Loop invariant

(Do not confuse with class invariant)

Property that is:

- Satisfied after initialization (from clause)
- Preserved by every loop iteration (loop clause) when executed with the exit condition (until clause) not satisfied
The loop invariant

from
   fancy.start ; \texttt{Result := ""}

\textbf{invariant}
   \begin{align*}
   & \texttt{fancy.index} \geq 1 \\
   & \texttt{fancy.index} \leq \texttt{fancy.count} + 1
   \end{align*}

\texttt{-- Result is the alphabetically highest of the}
\texttt{-- names of previous stations}

until
   fancy.after

\textbf{loop}
   \begin{align*}
   & \texttt{Result := greater (Result, fancy.item.name)}
   \end{align*}

   fancy.forth

end
The loop invariant (better)

```
from
    fancy.start; Result := ""
invariant
    index >= 1
    index <= count + 1
    -- If there are any previous stations, Result is the alphabetically highest of their names
until
    fancy.after
loop
    Result := greater (Result, fancy.item.name)

    fancy.forth
end
```
Loop as approximation strategy

\[ \text{Result} = \text{name}_1 \]

\[ \text{Result} = \text{Max} (\text{name}_1, \text{name}_2) \]

\[ \text{Result} = \text{Max} (\text{name}_1, \text{name}_2, ..., \text{name}_i) \]

\[ \text{Result} = \text{Max} (\text{name}_1, \text{name}_2, ..., \text{name}_i, ..., \text{name}_n) \]
In a function

\[ highest\_name : \text{STRING is} \]

\[ \text{-- Alphabetically greatest station name of line} \]

\[
\text{do} \]

\[
\text{from} \]

\[ \text{fancy.start : Result := ""} \]

\[
\text{until} \]

\[ \text{fancy.after} \]

\[
\text{loop} \]

\[ \text{Result := greater (Result, fancy.item.name)} \]

\[
\text{fancy.forth} \]

\[
\text{end} \]

\[
\text{end} \]
highest_name: STRING is
  -- Alphabetically greatest station name of line
  do
    from fancy.start; Result := ""
    until fancy.after
    loop
      Result := greater (Result, fancy.item.name)
    fancy.forth
  end
ensure
  Result /= Void and then not Result.empty
end
Loop as approximation strategy

Result = \textit{name}_1

Result = \textit{Max} (\textit{name}_1, \textit{name}_2)

Result = \textit{Max} (\textit{name}_1, \textit{name}_2, ..., \textit{name}_i)

Result = \textit{Max} (\textit{name}_1, \textit{name}_2, ..., \textit{name}_i, ..., \textit{name}_n)
The loop invariant

\begin{Verbatim}
from
  fancy.start; Result := ""

\textbf{invariant}
  fancy.index >= 1
  fancy.index <= fancy.count + 1
  -- Result is the alphabetically highest of the
  -- names of previous stations

until
  fancy.after

\textbf{loop}
  Result := greater (Result, fancy.item.name)

  fancy.forth

end
\end{Verbatim}
Loop invariant

(Do not confuse with class invariant)

Property that is:

- Satisfied after initialization (**from** clause)

- Preserved by every loop iteration (**loop** clause) when executed with the exit condition (**until** clause) **not** satisfied
The loop invariant

\[
\text{from} \\
\quad \text{fancy.start}; \ Result := ""
\]

\text{invariant}

\[
\quad \text{fancy.index} \geq 1 \\
\quad \text{fancy.index} \leq \text{fancy.count} + 1 \\
\quad \text{-- Result is highest of previous station names}
\]

\text{until}

\[
\quad \text{fancy.after}
\]

\text{loop}

\[
\quad \text{Result} := \text{greater}(\text{Result}, \text{fancy.item.name})
\]

\[
\quad \text{fancy.forth}
\]

\text{end}
The loop invariant (better)

\begin{verbatim}
from fancy.start ; Result := ""

invariant
  index >= 1
  index <= count + 1
  -- If there are any previous stations,
  -- Result is the alphabetically highest of their names

until fancy.after

loop
  Result := greater (Result, fancy.item.name)

  fancy.forth

end
\end{verbatim}
The effect of the loop

from
    fancy.start ; Result := ""

variant
    index >= 1
    index <= count + 1
    -- Result is highest of previous station names
until
    fancy.after
loop
    Result := greater (Result, fancy.item.name)

fancy.forth
end

At end: invariant and exit condition
• All stations visited (fancy.after)
• Result is highest of their names
Quiz: what’s the invariant?

```plaintext
xxx (a, b: INTEGER): INTEGER is
  -- ???????????????????????????????????????????
  require
    a > 0; b > 0
  local
    m, n: INTEGER
  do
    from
      m := a; n := b
    invariant
      -- "?????????"
    variant
      ????????
    until
      m = n
  loop
    if m > n then
      m := m - n
    else
      n := n - m
    end
  end
  Result := m
end
```

Intro. to Programming, lecture 8: Control structures I 56
Quiz: what’s the invariant?

```plaintext
xxx (a, b: INTEGER): INTEGER is
  -- ?????????????????????????????????????????
  require
    a > 0; b > 0
  local
    m, n : INTEGER
  do
    from
      m := a; n := b
    invariant
      -- "????????"
    variant
      ???????
  until
    m = n
  loop
    if m > n then
      m := m - n
    else
      n := n - m
    end
  end
  Result := m
end
```

Intro. to Programming, lecture 8: Control structures I 57
Quiz: what’s the invariant?

```plaintext
euclid (a, b: INTEGER): INTEGER is
    -- Greatest common divisor of a and b
require
    a > 0 ; b > 0
local
    m, n : INTEGER
do
    from
        m := a ; n := b
invariant
    -- "???????"  
variant
    ??????? 
until
    m = n
loop
    if m > n then
        m := m - n
    else
        n := n - m
    end
end
Result := m
end
```

Intro. to Programming, lecture 8: Control structures I 58
What we have seen

- The notion of algorithm
  - Basic properties
  - Difference with “program”
- The notion of control structure
- Correctness of an instruction

- Control structure: sequence
- Control structure: conditional
- Nesting, and how to avoid it
End of lecture 8