Introduction to Computers and Programming

Prof. I. K. Lundqvist

Recitation 1
Sept 11 2003

Some suggestions ...

- Problem solving
  - Feldman Case Study Format (the bank example)
- If-then-else statements
- Yesterdays PRS question (robot) / Truth table
- Binary, Hex, ASCII
- Little/Big Endian
- String manipulation

Case Study Format (p19)

1. Problem specification
2. Analysis
3. Design
4. Test plan
5. Implementation
6. Testing

1) Problem Specification

- A program is required which will ask the user for the amount of money (positive integer only) in a bank account. It will then ask for the amount of money (integers greater than zero) to be withdrawn.
- If the amount to be withdrawn is greater than the amount in the account, by more than $50, the program is to display a message that the transaction is refused, and the unchanged balance is displayed.
- If the amount of money to be withdrawn is less than or equal to the amount in the account, the transaction is accepted and the new balance in the account is displayed.
- If the amount to be withdrawn is greater than the amount in the account, by up to $50, the program is to accept the transaction and display the new balance, with a warning that the account is overdrawn.

2) Analysis

- Determine what you are asked to do:

1. Interact with user via text interface

   – Enter balance of the account **100**
   Enter the withdrawal **50**
   Accepted. Balance is 50

   Enter balance of the account **76**
   Enter the withdrawal **150**
   Refused! Balance is 76

   Enter balance of the account **50**
   Enter the withdrawal **75**
   Overdraft! Balance is -25

2) Analysis

- Determine what you are asked to do:

2. Act differently depending on balance in account after withdrawn:

Balance after withdrawal	Action
>= 0	Accept withdrawal
>= -50 and < 0	Overdraft
< -50	Refuse withdrawal

2) Analysis

- Data Requirements and Formulas
  - Problem Constant

```
Overdraft_Limit : constant Integer := -50;
Zero : constant Integer := 0;
```

  - Problem inputs

```
Balance      -- balance on account
Withdrawal -- amount to withdraw from account
```

  - Problem outputs

```
Resulting_Balance -- Balance after withdrawal
```

  - Formulas or relations

```
Resulting balance = Balance – withdrawal
```

3) Design

- Having listed the problem inputs and outputs, we can now list the steps necessary to solve the problem
- The Algorithm -- First try:
  1. Get balance and withdrawal
  2. Calculate resulting balance
  3. Is new balance
     - >= zero
     - >= -50 and < 0
     - or < -50 ?

3) Design

- The Algorithm -- Refinement:

1. Get balance and withdrawal
    1. Get balance
    2. Get withdrawal
2. Calculate resulting balance
    1. New balance = old balance – withdrawal
3. **If** new balance is >= zero
   then
    1. Indicate transaction accepted

   else if new balance between zero and overdraft limit
    2. Indicate overdraft is used

   else
    3. Indicate transaction rejected

4) Test Plan

- Cases that need to be tested are:
  - Balance = -40
    - Withdrawal = 5, 10, 11
  - Balance = 0
    - Withdrawal = 5, 50, 51
  - Balance = 20
    - Withdrawal = 20, 70, 71

5) Implementation

- Start with a basic Ada framework
- To write the final program, you must:
  - Convert the refined steps to Ada
  - Write Ada code for the unrefined steps
  - Add necessary context clauses for I/O
  - Delete the `NULL;` statement
  - Remove the step numbers from the comments
- Bank_Framework.adb

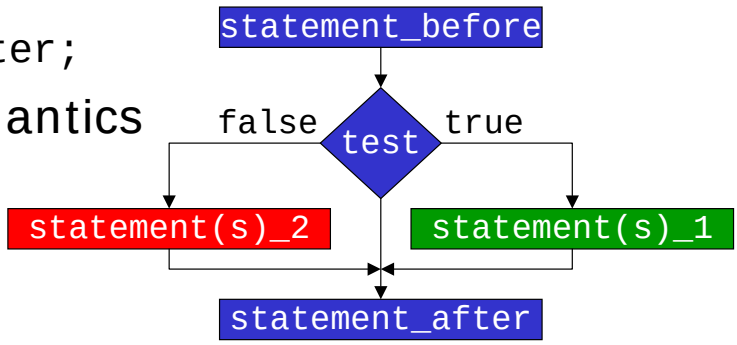
6) Testing

Balance	Withdrawal	Result
-40	5	OK
-40	10	OK
-40	11	OK
0	5	
0	50	
0	51	
20	20	
20	70	
20	71	

BACK

if-then-else Statements

- Statement form
  - statement_before;
    if test **then**
      statement(s)_1;
    else
      statement(s)_2;
    end if;
    statement_after;
- Statement semantics



Multiple Selections

- Statement form
  - statement_before;
    if test_1 **then**
      statement(s)_1;
    elsif test_2 **then**
      statement(s)_2;
    else
      statement(s)_3;
    end if;
    statement_after;

BACK

Yesterdays Robbie Event

- For the given input, which way will the robot behave?
  1. Go back once and turn left
  2. Turn right twice
  3. Go back twice the distance and turn right

Variable Name	Value
Hit_Left_Bumper	1
Hit_Right_Bumper	1
Driving_Left	1
Driving_Right	0
Control	1

```
if (Hit_Left_Bumper = 1) and (Hit_Right_Bumper = 1) then
        if (Control = 1) then
            S_1 { go back twice the distance
                  set Control to 0;
        else
            S_2 { go back normal distance
                  set Control to 1;
        end if;
end if;

if (Control = 1) then
        if (Driving_Left = 1) then
                turn left again;
                Driving_Left = 0;
        else
                turn right twice;
                set Driving_Right to 1;
        end if;
else
        if (Driving_Left = 0) then
                turn left;
                set Driving_Left to 1;
        else
                turn right;
                set Driving_Right to 1;
        end if;
end if;
```

test_1	test_2	test_3	test_1 **and** test_2	s_1	s_2
F	F	F			
F	F	T			
F	T	F			
F	T	T			
T	F	F			
T	F	T			
T	T	F			
T	T	T			

```
if (Hit_Left_Bumper = 1) and (Hit_Right_Bumper = 1) then
        if (Control = 1) then
            S_1 { go back twice the distance
                  set Control to 0;
        else
            S_2 { go back normal distance
                  set Control to 1;
        end if;
end if;

if (Control = 1) then
        if (Driving_Left = 1) then
                turn left again;
                Driving_Left = 0;
        else
                turn right twice;
                set Driving_Right to 1;
        end if;
else
        if (Driving_Left = 0) then
                turn left;
                set Driving_Left to 1;
        else
                turn right;
                set Driving_Right to 1;
        end if;
end if;
```

test_1	test_2	test_3	test_1 **and** test_2	s_1	s_2
F	F	F	F		
F	F	T	F		
F	T	F	F		
F	T	T	F		
T	F	F	F		
T	F	T	F		
T	T	F	T		
T	T	T	T	*	

BACK

Little/Big-Endian

- 00000000 00000000 00000100 00000001

Address	Big-Endian repr. of 1025	Little-Endian repr. of 1025
00	0000 0000	0000 0001
01	0000 0000	0000 0100
02	0000 0100	0000 0000
03	0000 0001	0000 0000

BACK

ASCII

BACK

ASCII	Hex	Symbol
0	0	NUL
1	1	SOH
2	2	STX
3	3	ETX
4	4	EOT
5	5	ENQ
6	6	ACK
7	7	BEL
8	8	BS
9	9	TAB
10	A	LF
11	B	VT
12	C	FF
13	D	CR
14	E	SO
15	F	SI

...

ASCII	Hex	Symbol
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

...

ASCII	Hex	Symbol
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o

01101000 01100101 01101100 01101100 01101111

Data types
String type

- Used when representing a sequence of characters as a single unit of data
  - How many characters?
  - String (1 .. Maxlen);
  - Example:

```
Max_Str_Length     : constant := 26;
Alphabet, Response:String(1..Max_Str_Length);
```

String Operations

- Assignment

```
Alphabet := “abcdefghijklmnopqrstuvwxyz”
Response := Alphabet;
```

- Concatenation (&)

```
Alphabet(1..3) & Alphabet(26..26)

Put(Item => “The alphabet is “ &
             Alphabet & “.”);
```

Sub-strings

- Individual character: specify position
  - `alphabet(10)  'j'`
    `alphabet(17)  'q'`
- Slice: specify range of positions
  - `alphabet(20..23)  "tuvw"`
    `alphabet(4..9)  "defghi"`
- Assign to compatible slice
  - `response(1..4) := "FRED";`
    `response "FREDefghijklmnopqrstuvwxyz"`

String I/O

- Text_Io
  - Output: Put, Put_Line
  - Get
    - Exact length needed
    - `Get(Item => A_String);`
  - Get_Line
    - Variable length accepted
    - Returns string and length
    - `Get_Line(Item => A_String, Last => N);`

BACK