Introduction to Computers and Programming

Prof. I. K. Lundqvist

Reading: B pp. 20-46 ; FK pp. 157-165, 245-255

Lecture 4
Sept 10 2003

Recap (1/3)

Context Clause `with Ada.Text_Io;`	Indicates that package Ada.Text_Io is used by the program
Program Heading `procedure Unified is`	Identifies Unified as the name of the program

Constant declaration
`Tax  : constant Float := 17.00;`
`Star : constant Character := '*';`
	Associates the constant, Tax, with the Float value 17.00

Variable declaration
`X : Float;`
`Y : Integer := 42;`
	Declares a variable object named X for storage of Integer values

Recap (2/3)

Assignment statement `Distance := Speed * Time;`	Computes the product of Speed and Time and assigns it to Distance

Input Statements
`Ada.Text_Io.Get`
`(Item =>Initial);`
	Enters data into the character variable Initial

Input Statements
`Ada.`**`Integer`**`_Text_Io.Get`
`(Item => Age);`
	… into the integer variable Age

Input Statements
`Ada.`**`Float`**`_Text_Io.Get`
`(Item => PayRate);`
	… into the float variable PayRate

Recap (3/3)

Output Statements `Ada.Text_Io.Put (Item =>Initial);`	Displays the value of the character variable Initial

Output Statements
`Ada.`**`Integer`**`_Text_Io.Put`
`(Item =>HowMany, Width=>3);`
	… integer variable HowMany, using five columns on the display

Output Statements
`Ada.`**`Float`**`_Text_Io.Put`
`(Item => GrossPay,`
`Fore => 4,`
`Aft => 2,`
`Exp => 0);`
	… float variable GrossPay using four columns before the decimal point and two columns after the decimal point

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

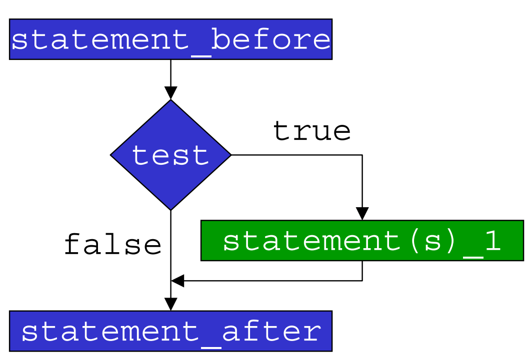
- Text_Io
  - Output: Put, Put_Line
  - Get
    - Exact length needed
    - `Get(Item => A_String);`
  - Get_Line
    - Variable length accepted
    - Returns string and length
    - `Get_Line(Item => A_String, Last => N);`

Control Structures
Selection statements

- Ada provides two types of selection statements
  - IF statements
    - **if-then**, when a single action might be done
    - **if-then-else**, to decide between two possible actions
    - **if-then-elsif**, to decide between multiple actions
  - Case statements, also for deciding between multiple actions

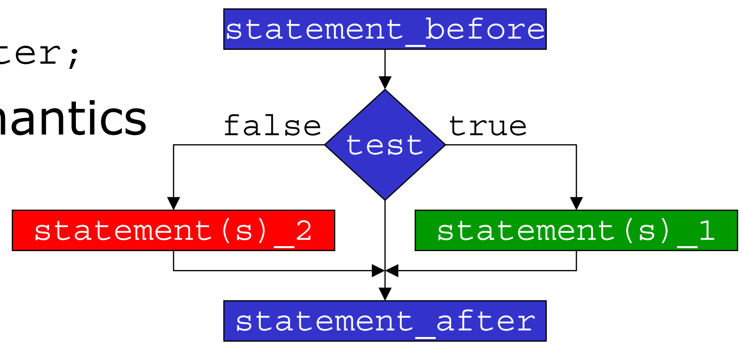
if-then Statements

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


    ```

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;
    ```

if_then_elsif Example (0/5)

- Resulting program of the following example was distributed in class today:

bank.adb

if_then_elsif Example (1/5)

- 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.

if_then_elsif Example (2/5)

- Decision table
  - A *multiple alternative if* may often be summarized by a **decision table** listing the alternatives

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

if_then_elsif Example (3/5)

- Alternative user interfaces
  - 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

if_then_elsif Example (4/5)

- Algorithm
  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

if_then_elsif Example (5/5)

- Data design

NAME	TYPE	Notes
Overdraft_Limit	Integer	-50 (for ease of change)
Zero	Integer	0 (for readability only)
Balance	Integer	Balance in the account
Withdrawal	Integer	Amount requested by user
Resulting_Balance	Integer	Balance after withdrawal

Conditions

- NOT

NOT(TRUE)	FALSE
NOT(FALSE)	TRUE

- OR

F or F	F
F or T	T
T or F	T
T or T	T

- AND

F and F	F
F and T	F
T and F	F
T and T	T

- XOR

F xor F	F
F xor T	T
T xor F	T
T xor T	F

Conditions Examples

- `(age < 18) or (sex = 'F')`
- `not ( (age >= 18) and (sex = 'M'))`
- `((age >= 60) and (sex = 'F')) or ((age >= 65) and (sex = 'M'))`
- Writing conditions and their associated actions correctly can be tricky. Truth tables can help you make sure the conditions and associated actions are correct.

Truth Tables

- Nested if statements

```
– if test_1 then
    if test_2 then
      statement(s)_1;
    else
      statement(s)_2;
    end if;
  else
    if test_3 then
      statement(s)_3;
    else
      statement(s)_4;
    end if;
  end if;
```

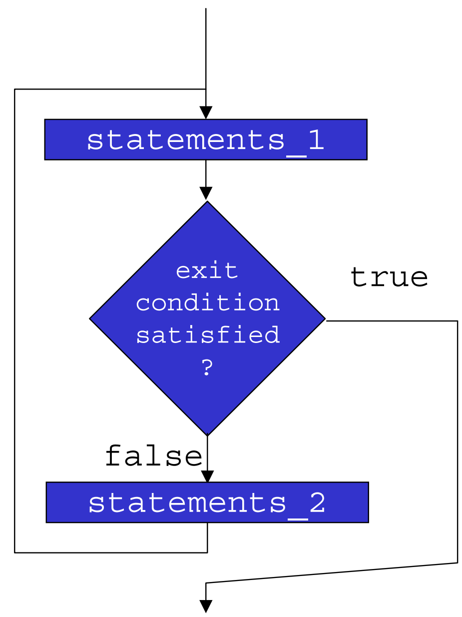
test_1	test_2	test_3	s_1	s_2	s_3	s_4
F	F	F				*
F	F	T			*	
F	T	F				*
F	T	T			*	
T	F	F		*		
T	F	T		*		
T	T	F	*			
T	T	T	*			

Control Structures
Loop Statements

- **Definite iteration** is where the set of actions is performed a known number of times. The number might be determined by the program specification, or it might not be known until the program is executing, just before starting the iteration.
  - Ada provides the **FOR statement** for definite iteration.
- **Indefinite iteration** is where the set of actions is performed a unknown number of times. The number is determined during execution of the loop.
  - Ada provides the **WHILE statement** and general **LOOP statement** for indefinite iteration.

General Loop Statements

- **loop**
    statements_1;
    exit when test;
    statements_2;
  end loop;



CQ

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

Bits, Nibbles, Bytes

- Bit (binary digit)
  - Two symbols: 0 / 1, false / true, …
- Byte
  - Collection of bits, usually 8 bits.
  - Always atomic, i.e., the smallest addressable unit
- Nibble
  - Half a byte, 4 bits.
  - More formally called a hex digit

Hexadecimal

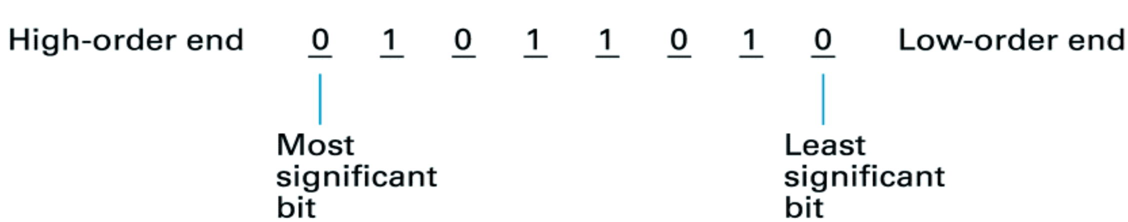
- Base 16 numeral system using symbols 0-9 and A-F
- Easy mapping from four bits to a single hex digit
- Can represent every Byte as two consecutive hexadecimal digits.

Hexadecimal

bin	hex	dec
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
	…	
1001	9	9
1010	A	10
1011	B	11
	…	
1111	F	15

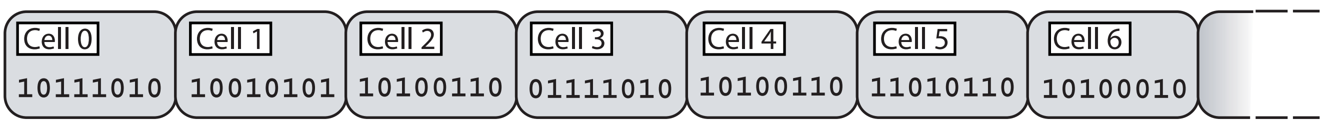
Main Memory

- Main memory arranged in manageable units called: Cells
  - Typically 8 bits: Byte



Memory Cells

- Each cell is assigned a unique name, called its address
- Stored data can be accessed in random order
- Read/ write



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

Memory Capacity

- Main memory systems usually has total number of cells as a power of two

Name	Abbr	Factor	SI size
kilo	K	$2^{10} = 1024$	$10^3 = 1000$
mega	M	$2^{20} = 1\ 048\ 576$	$10^6 = 1\ 000\ 000$
giga	G	$2^{30} = 1\ 073\ 741\ 824$	10^9
tera	T	$2^{40} = 1\ 099\ 511\ 627\ 776$	10^{12}
…			

Information as Bit Patterns

- Representing text, numeric values, images, sound
- Text
  - (Extended) ASCII (American Standard Code for Information Interchange)

Character on the screen	Binary value used to process it	Character on the screen	Binary value used to process it
1	0110001	A	1000001
2	0110010	B	1000010
3	0110011	C	1000011
4	0110100	D	1000100
5	0110101	E	1000101

  - EBCDIC (Extended Binary Coded Decimal interchange Code)
  - Unicode
  - ISO standards

ASCII

ASCII	Hex	Symbol		ASCII	Hex	Symbol		ASCII	Hex	Symbol
0	0	NUL		**48**	30	0		**96**	60	`
1	1	SOH		**49**	31	1		**97**	61	a
2	2	STX		**50**	32	2		**98**	62	b
3	3	ETX		**51**	33	3		**99**	63	c
4	4	EOT		**52**	34	4		**100**	64	d
5	5	ENQ		**53**	35	5		**101**	65	e
6	6	ACK	...	**54**	36	6	...	**102**	66	f
7	7	BEL		**55**	37	7		**103**	67	g
8	8	BS		**56**	38	8		**104**	68	h
9	9	TAB		**57**	39	9		**105**	69	i
10	A	LF		**58**	3A	:		**106**	6A	j
11	B	VT		**59**	3B	;		**107**	6B	k
12	C	FF		**60**	3C	<		**108**	6C	l
13	D	CR		**61**	3D	=		**109**	6D	m
14	E	SO		**62**	3E	>		**110**	6E	n
15	F	SI		**63**	3F	?		**111**	6F	o

01101000 01100101 01101100 01101100 01101111

Numeric Values

- Storing the value of 25_{10} using ASCII:

  00110010 00110101

- Binary notation: $00000000\ 000011001_2$

Base ten system

Representation	3	7	5
Position's quantity	Hundred	Ten	One

1. Binary place	2^5	2^4	2^3	2^2	2^1	2^0
2. Position's quantity	32	16	8	4	2	1
3. Example binary pattern	1	0	1	1	0	1

4. Total (2. x 3.) 32 + 0 + 8 + 4 + 0 + 1 = 45

Base two system

Representation	1	0	1	1
Position's quantity	Eight	Four	Two	One

Finding Binary Representation of Large Values

1. Divide the value by 2 and record the remainder
2. As long as the quotient obtained is not 0, continue to divide the newest quotient by 2 and record the remainder
3. Now that a quotient of 0 has been obtained, the binary representation of the original value consists of the remainders listed from right to left in the order they were recorded