Programming Paradigms

these slides contain advanced material and are optional

Programming paradigms/languages

- Machine languages
- Procedural
- Object-oriented
- Prototype-based
- Functional
- Visual
- Logic
- Hardware
- Esoteric
- Multi-paradigm

Resources

- Code examples taken from

  http://99-bottles-of-beer.net/

- Wikipedia

  http://en.wikipedia.org/wiki/Programming_paradigm

  http://en.wikipedia.org/wiki/List_of_programming_languages_by_category

  http://en.wikipedia.org/wiki/List_of_multi-paradigm_programming_languages

Machine languages

- Low-level
  - Direct CPU instructions
  - Direct access to CPU registers
  - Direct access to memory
- Easy to compile
  - Each instruction has a bit-representation
  - Single-pass translation

- Example: x86 Assembler

  http://99-bottles-of-beer.net/language-assembler-(intel-x86)-1144.html

x86 Assembler

```
segment .text
; this function converts integer in range 0-99 to string
_integer_to_string:
        mov     eax, dword [esp + 08h]    ; get the vavlue
        mov     ecx, 10                   ;
        sub     edx, edx
        div     ecx                       ; divide it by 10
        mov     ecx, dword [esp + 04h]    ; get the output offset
        test    eax, eax                  ; is greater than 9
        jz      .skip_first_digit         ; skip saving 0 char if no
        add     al, 030h                  ; convert number to ascii char
        mov     byte [ecx], al            ; save
        inc     ecx                       ; increase pointer
        jmp     .dont_test_second_digit   ;
    .skip_first_digit:                    ; only if less then 10
        test    edx, edx
        jz      .skip_second_digit
    .dont_test_second_digit:              ; if it was greater than 10
        add     dl, 030h                  ; than second digit must by
        mov     byte [ecx], dl            ; written at no condition
        inc     ecx
    .skip_second_digit:                   ; only skip if value was 0
        mov     byte [ecx], ah            ; save the null ending char
        retn    4                         ; ret and restore stack
```

Procedural

- Structured programming
  - Procedures
  - Data global or per module
  - Control structures: loops, conditionals

- Example: Pascal

  http://99-bottles-of-beer.net/language-turbo-pascal-470.html

Pascal

```
program Bottles;

uses wincrt;
var b: byte;

function plural(anz_flaschen: byte): string;
begin
  if anz_flaschen <> 1
    then plural:= 's'
    else plural:= ''
end; {plural}

begin
  screensize.y:= 1 + 99 * 5;
  inactivetitle:= ' 99 Bottles of beer ';
  initwincrt;
  for b:=99 downto 1 do
    begin
      writeln(b :2, ' bottle' + plural(b) + ' of beer on the wall, ');
      writeln(b :2, ' bottle' + plural(b) + ' of beer.');
      writeln('Take one down, pass it around,');
      writeln((b-1) :2, ' bottle' + plural(b-1) + ' of beer on the wall.');
      writeln
    end
end. {Bottles}
```

Object-oriented: class-based

- Classes as operation abstraction
- Objects as data abstraction
- Inheritance
- Dynamic binding

- Example: Eiffel

  http://99-bottles-of-beer.net/language-eiffel-231.html

Eiffel

```
class BEER
create
   make
feature

   shelf: SHELF

   make
      do
         from
            create shelf.make (99)
         until
            shelf.empty
         loop
            io.put_string (shelf.description)
            shelf.remove
            io.put_string ("Take one down, pass it all around%N%N")
         end
         io.put_string (shelf.description)
         io.put_string ("Go to the store and buy some more%N%N")
         shelf.make (99)
         io.put_string (shelf.description)
      end

end
```

Object-oriented: prototype-based

- No class definitions
- Data and functions are added to objects
- Objects are cloned to create new objects

- Example: JavaScript

  http://99-bottles-of-beer.net/language-eiffel-231.html

JavaScript

```
var Song = function(){};
//add methods to the prototype, to affect the instances of the class Song
Song.prototype = {
        map: function( src, fn ){
                var
                        mapped = [ ], //will hold the mapped items
                        pos = src.length; //holds the actual index
                while( pos-- )
                        mapped[pos] = fn.call( this, src[pos], pos );
                return mapped;
        },
        bottle:function( left ){
                switch( left ){
                        case 0: return 'no more bottles';
                        case 1: return '1 bottle';
                        default: return left + ' bottles';
                }
        },
        buy:function( amount ){ this.bottles = Array(amount+1); },
...
};
var bottlesSong = new Song();
bottlesSong.buy( 99 );
var lyrics = bottlesSong.sing( '<br />' );
document.body.innerHTML = lyrics;
```

Logic

- Declare facts and rules
- Ask questions
- Automatically resolved
  - SLD resolution
  - Backtracking

- Example: Prolog

  http://99-bottles-of-beer.net/language-prolog-1114.html

Prolog

```
wall_capacity(99).

wait(_) :- true.

report_bottles(0) :- write('no more bottles of beer'), !.
report_bottles(X) :- write(X), write(' bottle'),
                     (X = 1 -> true ; write('s')), write(' of beer').

report_wall(0, FirstLine) :- (FirstLine = true -> write('No ') ; write('no ')),
                             report_bottles('more'), write(' on the wall'), !.
report_wall(X, _) :- report_bottles(X), write(' on the wall').

sing_verse(0) :- !, report_wall('No more', true), write(', '),
                 report_bottles('no more'), write('.'),
                 nl, write('Go to the store and buy some more, '),
                 wall_capacity(NewBottles), report_wall(NewBottles, false),
                 write('.'), nl.
sing_verse(X) :- report_wall(X, true), write(', '),
                 report_bottles(X), write('.'), nl,
                 write('Take one down and pass it around, '),
                 Y is X - 1, report_wall(Y, false), write('.'), nl, nl,
                 wait(5), sing_verse(Y).

:- wall_capacity(Bottles), sing_verse(Bottles).
```

Functional

- Stateless & Side-effect free
- More like mathematical functions
- Higher-order functions
  - Functions as arguments and results

- Example: Haskell

  http://99-bottles-of-beer.net/language-haskell-1613.html

Haskell

```
bottles :: Int -> String
bottles n
  |n == 0 = "no more bottles"
  |n == 1 = "1 bottle"
  |n >  1 = show n ++ " bottles"

verse :: Int -> String
verse n
  |n == 0 = "No more bottles of beer on the wall, no more bottles ..."
         ++ "Go to the store and buy some more, 99 bottles of beer ..."
  |n > 0  = bottles n ++ " of beer on ..., " ++ bottles n ++ " of beer.\n"
         ++ "Take one down and pass it around, " ++ bottles (n-1)
         ++ " of beer on the wall.\n"

main  = mapM (putStrLn . verse) [99,98..0]
```

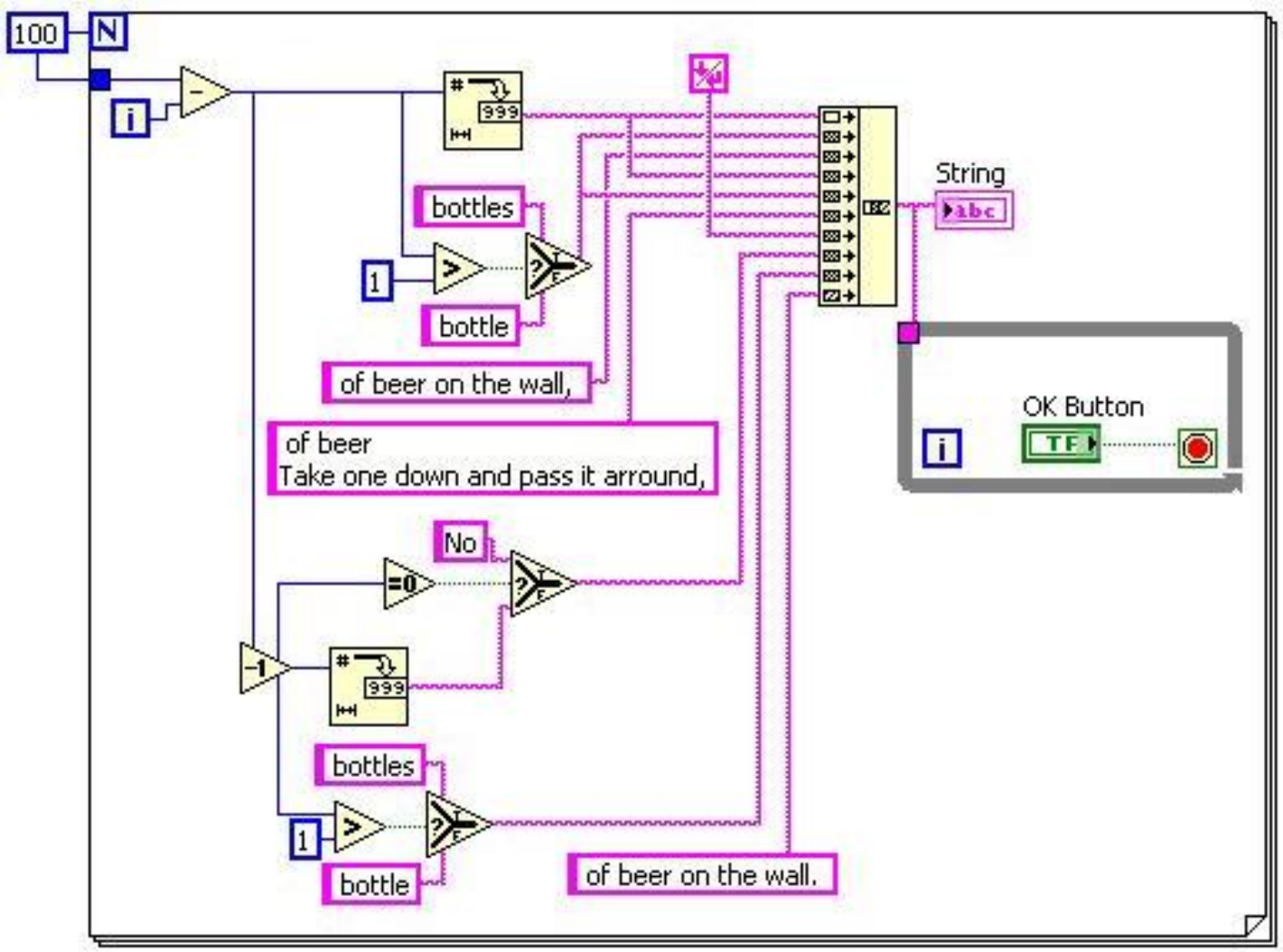
Visual

- Program represented by diagram
- Possible to visualize program execution / data flow

- Example: LabView

  http://99-bottles-of-beer.net/language-labview-729.html

LabView


100
N
i
999
bottles
1
bottle
of beer on the wall,
of beer
Take one down and pass it arround,
String
abc
OK Button
TF
No
=0
-1
bottles
1
bottle
of beer on the wall.

Hardware

- Limited instructions
  - Signal input/output
  - Choice
  - Limited loops (unrolling)

- „Compiled“ to hardware

- Example: VHDL

  http://99-bottles-of-beer.net/language-vhdl-168.html

VHDL

```
entity beer_song is
  port(bottles: out integer; words: out string(1 to 28); start_singing: in boolean);
end beer_song;
architecture silly of beer_song is
begin
    lets_sing: process
    begin
        wait on start_singing until start singing;
        for index_bottles in 99 downto 1 loop
                bottles <= index_bottles;
                words <= "bottles of beer on the wall,";
                wait for 5 sec;
                bottles <= index_bottles;
                words <= "bottles of beer,           ";
                wait for 5 sec;
                words <= "take one down,             ";
                wait for 5 sec;
                words <= "pass it around,            ";
                wait for 5 sec;
                bottles <= index_bottles - 1;
                words <= "bottles of beer on the wall."
                wait for 5 sec.
        end loop;
        assert false report "No more beer!" severity warning;
    end process lets_sing;
end silly;
```

Esoteric

- Whatever you can imagine

- Example: Brainfuck
  http://99-bottles-of-beer.net/language-brainfuck-1539.html

- Example: Whitespace
  http://99-bottles-of-beer.net/language-whitespace-154.html

Brainfuck

```
# Set beer counter to 99
>>>>>>>>>
>++++++++++[-<++++++++++>]<-
<<<<<<<<<<
# Create output registers
++++++++++[->++++>++++>++++>++++<<<<]add 0x28 to all from (1) to (4)
++++++++[->>>++++++++>++++++++<<<<]  add 0x40 to all from (3) and (4)
++++[->>>>++++<<<<]                        add 0x10 to (4)
++++++++++                                 set (0) to LF
>--------                            set (1) to SP
>++++                                       set (2) to comma
>>>>>>>                     go to beer counter (9)
[
         <<<<
         +++       state 1 in (5)
         >+        state 2 in (6)
         >++       state 3 in (7)
         <<        go to (5)
         [
                   >>>>              go to (9)
                   [
                            [->+>+<<]>>[-<<+>>]<[>++++++++++[->>+>+<<<]
                            <[>>>[-<<<-[>]>>>>[<[>]>[----------->>]<++++
                            ++[-<++++++++>]<<[<->[->-<]]>->>>[>]+[<]<<[
                            ->>>[>]<+[<]<<]<>>]<<]<+>>[->+<<+>]>[-<+>]<
                            <<<<]>>[-<<+>>]<<]>[-]>>>>>>[>]<[.[-]<]
                            <<<<<<
```

Whitespace

Multi-paradigm

- Most languages combine different paradigms
- Java
  - imperative/procedural, generic, reflective, object-oriented (class-based)
- Eiffel
  - imperative/procedural, generic, object-oriented (class-based), concurrent (SCOOP)
- Oz
  - concurrent, constraint, dataflow, distributed, functional (evaluation: eager, lazy), imperative, logic, object-oriented (class-based)