

Imperative Programming Part I

1



Programming Languages

- Different kinds of languages
 - Functional (lisp, scheme, ...)
 - Database (SQL, ...)
 - Logic (prolog, ...)
 - Assembly (Sparc, ...)
 - Imperative (Java, C, C++, Pascal, ...)

2



Imperative

imperative - expressing command;
commanding, peremptory; urgent;
obligatory.

- An imperative program is a sequence of commands.

3



Some commands

- Assume a simple robot can obey the following program commands:
 - *forward* - move 50 cm forward.
 - *left* - turn left.
 - *right* - turn right.
- In Java, the robot is instructed to move using a statement like:
 `robot.forward() ;`

4



Robot program: general structure

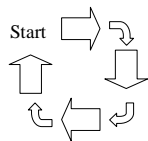
```
public class Robot
{
    public static void forward()
    {
        ...
    }
    public static void left()
    {
        ...
    }
    public static void right()
    {
        ...
    }
    public static void main(String[] args)
    {
        ...
    }
}
```

5



A very simple program

```
robot.forward() ;
robot.right() ;
robot.forward() ;
robot.right() ;
robot.forward() ;
robot.right() ;
robot.forward() ;
```



Note the punctuation.
Each command is *terminated*
by a semi-colon.

6



How do we know what the robot does?

- Each command has a well defined meaning.
- Each statement is carried out in the order given by the written sequence.

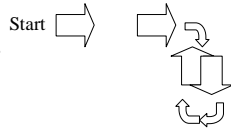
Of course, we assume there are no unexpected problems, like a hole in the floor!

7



Same commands, different sequence

```
robot.forward();  
robot.forward();  
robot.right();  
robot.forward();  
robot.right();  
robot.forward();  
robot.forward();
```



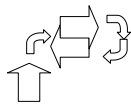
End up in a different place.

8



Same program, different starting state

```
robot.forward();  
robot.forward();  
robot.right();  
robot.forward();  
robot.right();  
robot.forward();  
robot.forward();
```



Start

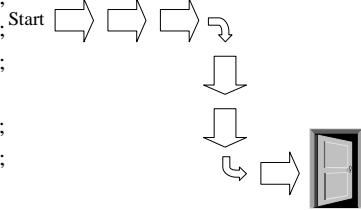
Facing a different direction at start.

9



Going to the door

```
robot.forward() ;  
robot.forward() ;  
robot.forward() ;  
robot.right() ;  
robot.forward() ;  
robot.forward() ;  
robot.left() ;  
robot.forward() ;
```



Easy - Problem solved??

10

Department of Computer Science



What if ?

- What if the robot starts off facing in a different direction?
- What if the robot is put in a different starting position or a different room?
- What if the door moves...

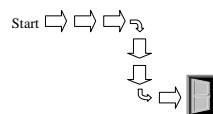
11

Department of Computer Science



The program context

- The program only solves the problem (getting to the door) in very specific circumstances.
- Change the circumstances, rewrite the program...



12

Department of Computer Science



Too much hard work in this case!

- In fact, *you* have to do all the work of solving the problem.
- The robot just does exactly what you dictate.
- No way!
 - Let's get the computer to do some of the hard work.

13

Department of Computer Science



What we would really like

- A program that can guide the robot to the door in *any* room, starting in *any* position.
- A program that *doesn't* have to be changed for every occasion.

14

Department of Computer Science



Too hard - simplify

- Let's try to solve a simpler problem first.
- If we can solve that, hopefully we can learn more about solving the harder problem.

Strategy: Look for a sub-problem and solve it first.

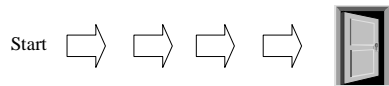
15

Department of Computer Science



First go

- Assume the robot is always facing in the direction of the door.
- Assume there are no obstacles between the robot and the door.



16

Department of Computer Science



Easy?

- Program is just:
`robot.forward() ;`
`robot.forward() ;`
`robot.forward() ;`
- But how many forwards?
- Depends where the robot starts from.
- We still have to change the program for every start position.

17

Department of Computer Science



Solution

- We need to be able to say: “Keep moving forward while you have not reached the door”.
- But we need additional commands to do this.

18

Department of Computer Science



A While Loop

```
while (not atDoor)
{
    robot.forward();
}
```

Loop Body

- Called a “loop” as it loops around!
- Keep looping while a condition is true.

19

Department of Computer Science



Syntax of the *while* statement

```
keyword while (not atDoor)
while
{
    robot.forward();
}
```

Boolean expression (true or false)

Loop Body (Sequence of instructions)

20

Department of Computer Science



Details

- *while* is a sequencing command providing *iteration*.
- *atDoor* is a test command that has a *boolean* value.
- *not* is a logical operator that is applied to a boolean value.

21

Department of Computer Science



A Result!

- We now have a program that will move the robot to the door, regardless of the starting position.
- Providing it is facing the door.
- Providing there are no obstacles.

22

Department of Computer Science



A Harder Problem

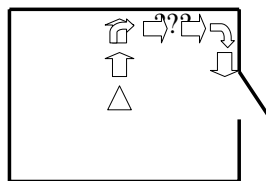
- We've solved our simple problem, so let's make it a bit harder and see what happens.
- Assume *robot* can start facing any direction.
- Assume *robot* is in a room with four walls, one door and no obstacles.
- We want the robot to find the door (not necessarily facing it)

23

Department of Computer Science



Brainstorming!!



24

Department of Computer Science



An idea!!

- Move forward until robot hits the wall.
- Turn right and follow wall.
- When the next wall is found, turn right and follow new wall.
- Stop when door is reached.

Now turn this into commands!

25

Department of Computer Science



Try another program

```
while (robot.noDoorNoWall())
{
  robot.forward() ;
}
robot.right() ;
while (robot.noDoorNoWall())
... Hmmm, how often do we do this?
```

26

Department of Computer Science



Rethink

```
while (not robot.atDoor())
{
  while (robot.noDoorNoWall())
  {
    robot.forward() ;
  }
  robot.right() ;
}
```

} A *nested* while loop.

27

Department of Computer Science



Wow!

- One new test command.
- And we have a program that will work in *any* empty room.
- (Providing all our assumptions remain true.)

28

Department of Computer Science



Obstacles

- An unexpected pile of bricks...



- Need to divert round the bricks.
- Any need to change the program?

29

Department of Computer Science



Choice

- Suppose the floor is covered in tiles of different colours.
- When the robot moves on to a red tile it plays a tune.
- When the robot moves on to a blue tile it beeps.
- When the robot move on to a green tile it remains silent.

30

Department of Computer Science



If...

- We want to say:
“If the tile is red, then play a tune”
“If the tile is blue, then beep”
“If the tile is green, be silent”
- A *selection* command is needed.
- Also need commands to test the tile colour and make noises.

31

Department of Computer Science



Let's Try...

```
while(robot.noDoorNoWall())  
{  
  robot.forward();  
  if (robot.isTileRed())  
  {  
    robot.playTune();  
  }  
  if (robot.isTileBlue())  
  {  
    robot.playBeep();  
  }  
}  
  
if (robot.isTileGreen())  
{  
  robot.playSilent();  
}
```

32

Department of Computer Science



Syntax of the *if* statement

```
keyword  if (robot.isTileRed())  
if       {  
         robot.playTune();  
         }  
         }  
         }
```

Boolean expression
(true or false)

If Body
(Sequence of instructions)

33

Department of Computer Science



Results

- We can now write a program to get the robot to find the door of a room.
- And play sounds as it moves around.
- We've discovered *iteration* and *selection* as being fundamentally necessary to get things working.

34

Department of Computer Science



Choosing our commands

- The commands we use need to be at the right level of detail.
- Too low level - too awkward to work with.
- Too high level - can't express the amount of detail needed.

35

Department of Computer Science



Abstraction

- *abstraction*:
 - A representation or model that includes the important, essential or distinguishing aspects of something while suppressing or ignoring less important, immaterial or diversionary details.
 - Removing distinctions to emphasise commonality.

36

Department of Computer Science



Choosing the right abstractions

- Our basic commands, selection and iteration are all abstractions of behaviour.
- They represent the lowest level of abstraction that we generally want to work with.
- (We can go lower - assembly language programming or even direct binary coding...)

37

Department of Computer Science



Creating new abstractions

- Programming relies heavily on using and creating abstractions.
- As programs get larger we have to create new abstractions to manage the huge amount of detail involved.

38

Department of Computer Science



Further thoughts...

- We have seen programs with the basic structure of:
Start → Do the work → Stop
Where we explicitly expect the program to run to a pre-determined conclusion.

Are all programs like that?

39

Department of Computer Science



Programs that stop?

- The robot “find the door” program may never stop...
 - but that would be a flaw that requires the program to be fixed.
- What about the drawing programs? How do they stop?
- What about a word processor?
- Or the control system in a car?

40

Department of Computer Science



Run for ever...

- In fact, many kinds of programs are designed to run continuously until the *user explicitly stops* them.
- If the program stops any other way something has gone wrong...

41

Department of Computer Science



The main loop

- Consider this overall program structure:

```
while (true)
{
  doWork ;
  if (userQuits)
  {
    stop ;
  }
}
```

42

Department of Computer Science



Event driven

- Loop forever, responding to *events*.
 - Events are mouse clicks, key presses, timer and so on.
- ```
while (true)
{
 waitForEvent ;
 handleEvent ;
 if (userQuits)
 {
 stop ;
 }
}
```

43

Department of Computer Science



---

---

---

---

---

---

---

---

## And the drawing programs?

- Your drawing programs are actually event driven, with a main loop.
- You don't see that explicitly - it comes for free as part of the infrastructure of the Java system.

44

Department of Computer Science



---

---

---

---

---

---

---

---

## Stopping drawing programs

- To stop our drawing programs we cheat!!
- Ctrl-C causes the operating system to stop a program.
- Our programs simply don't (yet) provide a way for the user to issue a stop event.

45

Department of Computer Science



---

---

---

---

---

---

---

---

## The Graphical User Interface (GUI)

- All programs that use windows, the mouse and so on, have a GUI.
- These programs are all event driven, with a main loop.
- They are designed to keep running for ever until you tell them to stop.

46

Department of Computer Science



---

---

---

---

---

---

---

---

## Event loop means GUI?

- No. An event driven program doesn't need a GUI.
- Consider control systems (in a car, plane, lift, ATM machine).
- They run continuously, responding to events, until explicitly stopped.

47

Department of Computer Science



---

---

---

---

---

---

---

---

## Summary

- Sequence, iteration, selection.
- Problem solving by decomposing a large problem into simpler smaller problems.
- Starting to think about abstraction.
- The main loop and event driven programming.

48

Department of Computer Science



---

---

---

---

---

---

---

---