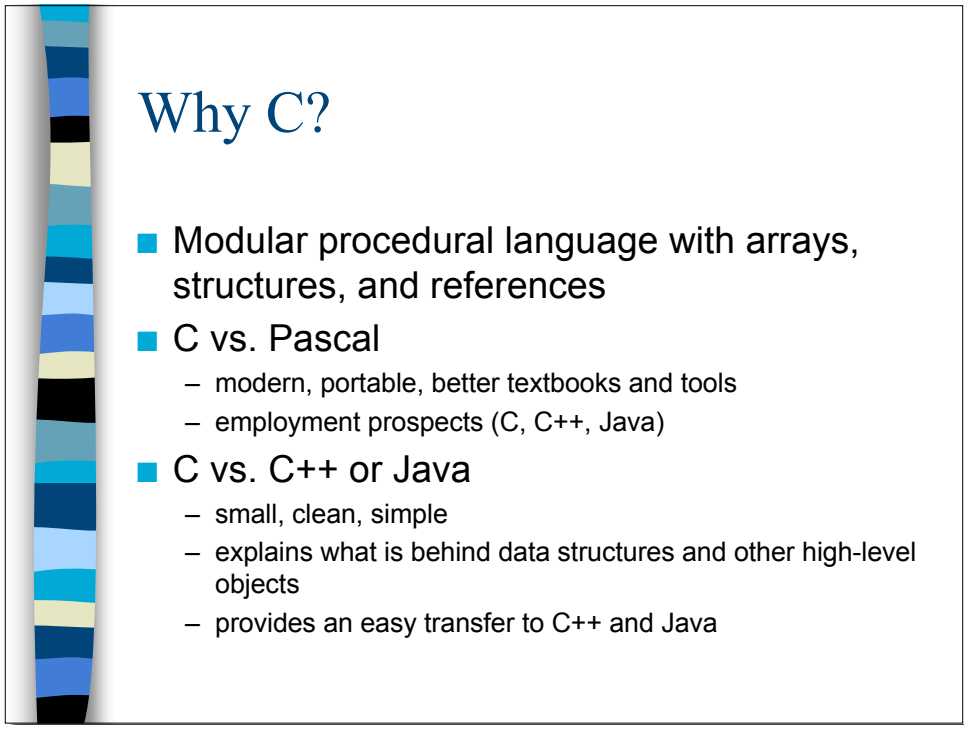
IS12 - Introduction to Programming

Lecture 7: Introduction to C

Peter Brusilovsky
http://www2.sis.pitt.edu/~peterb/0012-072/

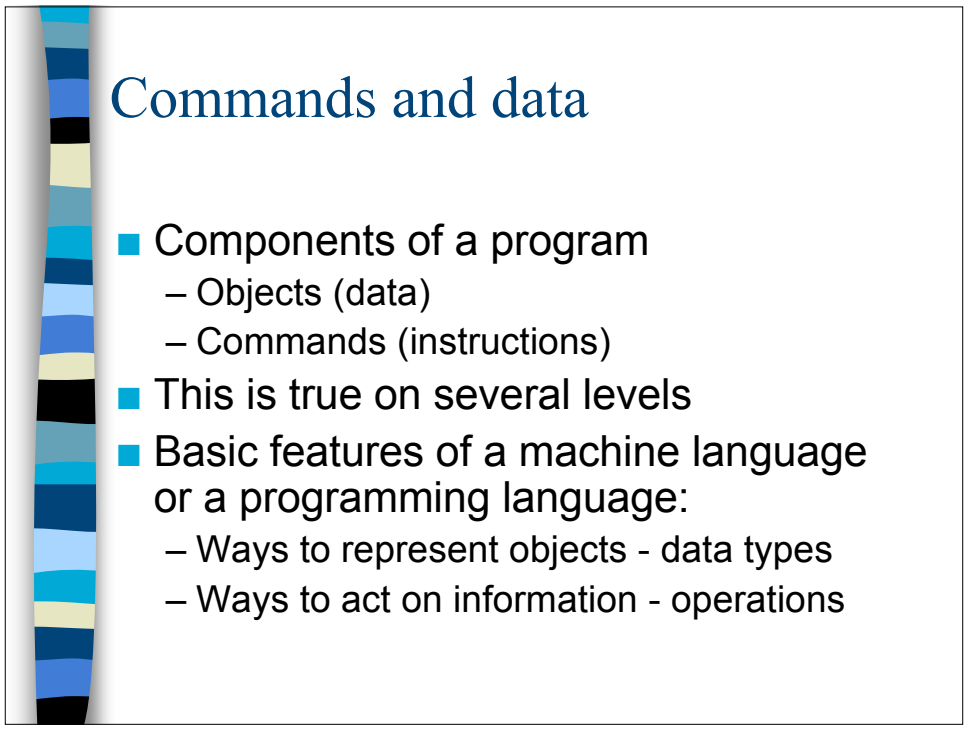
Lecture 7: Introduction to C. Overview

- C vs. other languages
- Information representation
- Let's start
- First C program (Hello, World!)
- Edit-compile-run loop again
- Errors and modifications
- Knowledge Tree as a great help for us



Why C?

- Modular procedural language with arrays, structures, and references
- C vs. Pascal
  - modern, portable, better textbooks and tools
  - employment prospects (C, C++, Java)
- C vs. C++ or Java
  - small, clean, simple
  - explains what is behind data structures and other high-level objects
  - provides an easy transfer to C++ and Java



Commands and data

- Components of a program
  - Objects (data)
  - Commands (instructions)
- This is true on several levels
- Basic features of a machine language or a programming language:
  - Ways to represent objects - data types
  - Ways to act on information - operations

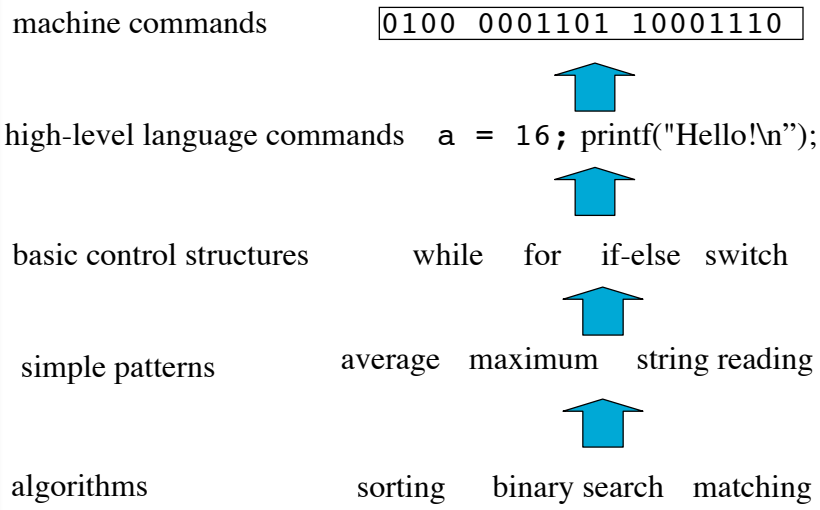
Karel vs. C

- Data
  - Karel - location, beepers, walls
  - C - numbers and symbols in memory
- Commands
  - Karel - move, turnleft
  - C - add, print…
- Karel operates in a visible word outside; C programs work invisibly inside

Information Representation

- Computer store information digitally in binary format
- Ultimately everything is ones and zeroes
  - characters, numbers
  - instructions (programs!!)
  - pictures, video
- Binary arithmetic E.g., 7 = 00000111
  99 = 01100011

From Commands to Algorithms



Learning C

- Be careful, read your programs!
  - The basic philosophy of C: "programmers know what they are doing" - K&R2, p.3
- Ask questions in CourseWeb forums
- Meet your instructor during the office hours
- Practice, practice, practice!
  - Run all examples, modify it
  - Solve problems, check yourself on quizzes
- Books: See Course Books page
- Tools: Editor+Compiler - See Course Tools page

Start Dev C++

- Start -> All Programs -> Programming Tools -> Bloodshed Dev C++ -> Dev C++
- Click New Program (blank page)
- Let's write, save, compile and run our first C program
- Continue prectice at home:
  - Install editor-compiler or IDE
  - Compile and run Hello, World! program
  - Experiment: print your name, make errors, etc.

Hello World Program

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n");
}
```

Dissection adapted From S. Pilachewski

Working with Command Prompt

1. Start->Run->cmd, or

Start -> All Programs -> Accessories ->Command Prompt

2. cd \
3. cd temp
4. mkdir IS12
5. cd IS12

Save and Run

- In Dev C++:
  - Save
  - Select My Computer-> Hard Disk (C:)
  - Compile
- In Command Prompt:
  - dir (lets you check that .exe is here)
  - name-of-the-program (runs the program)
- Alternatively run in Dev C++
  - system("PAUSE");

C Program Syntax

- Most C programs have the following (at minimum):

```
main ([program arguments])
{
    <one or more statements>
}
```

- Every program must have a “main” function. Note that C is case sensitive (Main is not the same as MAIN or main).

Hello World - dissected

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n");
}
```

- This is a *comment*. Comments are written not for computers but for humans. The computer will ignore everything between /* and */. Humans need comments to understand the program. (Why?)

Hello World - dissected

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n");
}
```

- *#include* is a *command* which tells the compiler that the standard input / output library will be used. Thus printf will be recognized as a standard output function

Hello World - main() Function

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n");
}
```

- Execution of a C program always begins at the ***main()*** function. Every C program must have one (only one) main() function.

Hello World - the Braces

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n");
}
```

- The open brace (**{**) marks the beginning of the function body, which is one or more program statements which perform some task.
- The closing brace (**}**) marks the end of the function body.

Hello World - the *printf* statement

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n");
}
```

- This *statement* is a function call to the *printf* function in the C Standard I/O Library. It displays the message which is the argument to the function. The **\n** denotes the newline. We can use printf because we told the compiler to use Standard I/O Library in *#include*

Hello World - the *semicolon*

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n");
}
```

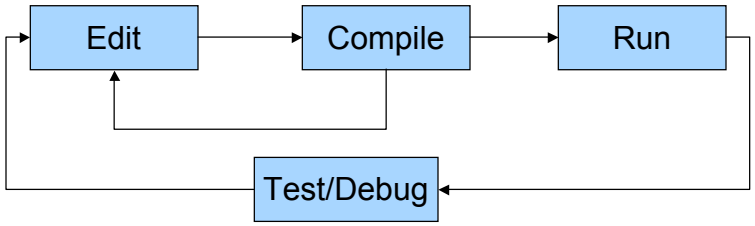
- The semicolon marks the *end* of a C program statement.

The edit-compile loop again

1. Edit program
2. Compile program
3. If there are errors, fix and go back to 1
   - you have got syntax error
   - fix and go back to 1
4. Run it
5. If it produce wrong results
   - you have got semantic error
   - find the source of the error (debug)
   - fix and go back to 1

The iterative nature of program design

The "programming in small" loop



Hello World - Syntax Error

```
/* This is our first program */
#include <stdio.h>

main()
{
     printf("Hello World!\n")
}
```

- The semicolon is missing
- What happens when we compile this?

Hello World - Experiment 2

```
/* This is our first program */
#include <stdio.h>

main()
{
     /* note the absence of \n */
     printf("Hello World");
     system("PAUSE");
}
```

- What happens when we run this?

Hello World - Experiment 3

```
/* This is our first program */
#include <stdio.h>

main()
{
     /* note the extra \n */
     printf("Hello\n World\n");
}
```

- What happens when we run this?