Software Project

Lecturers: Ran Caneti, Gideon Dror
Teaching assistants: Nathan Manor, Ben Riva

Emails: (canetti/benriva)@post.tau.ac.il
nathan.manor@gmail.com
gideon@mta.ac.il

http://www.cs.tau.ac.il/~roded/courses/soft-project10.html

These slides:
http://www2.mta.ac.il/~gideon/courses/c/slides/

A Book on C - ABC
Kelley/Pohl, Addison-Wesley
Fourth Edition
Course Structure

• The C programming language: 8-9 classes including 3 exercises (20%).

• Large project, in pairs (50%). 2 classes will be devoted to its presentation.

• Final exam (30%).
The C programming language

- B, initial versions of UNIX, 1970
- C, Dennis Ritchie, Bell Labs, 1972; overcomes B’s typeless representation; used for developing UNIX.
- Early 80’s - traditional C
- 1990 – the ANSI-C standard
- C++ (1980) and Java (1990’s)
C is powerful and efficient: direct memory management, some operators directly modify machine registers, manipulation of memory addresses. Efficient compiler.

C is compact: few reserved words, concise commands, powerful set of operators.

C is portable: code written on one machine can be easily moved to another.

C is modular and typed.

C is the native language of UNIX.

• Basis of C++ (and Java).
C vs. Java

• Java was developed based on C and inherited most of its syntax.

• Main difference: C is procedural and not object-oriented. Thus, no classes/interfaces; methods are called functions and do not belong to any object.

• Similar primitive types as in Java, but no boolean and string types;

• C is much less structured. It is easier to make mistakes and harder to recover
C vs. Java (cont.)

• Pointers instead of references. Allow direct access to memory and dereferencing (but less elegant).

• No garbage collection – need to manage memory explicitly.

• Preprocessor.

• Compilation to machine code – faster but some platform dependency; java is processed via an interpreter (JVM).
C vs. Java – small differences

• No for/in (iterating within a set)
• No function overloading
• Arrays are not objects – cannot apply methods to them (e.g. no cloning).
• Global variables.
• Variable declarations only in the beginning of a block.
• Composite types such as union and bitfield; typedef.
• Function pointers.
C Topics

• Lexical elements (Ch. 2)
• Fundamental data types (Ch. 3)
• Flow of control (Ch. 4)
• Functions (Ch. 5); Runtime environment
• Arrays, pointers and strings (Ch. 6); Dynamic matrix allocation (Ch. 12.6)
• Bitwise operators (Ch. 7)
• Preprocessor (Ch. 8)
• Structures and enumeration types (Chs. 9 & 7.5)
• Linked lists (Ch. 10)
• Input/output and files (Ch. 11)
Programming Environment

- Unix
- Make
Operating System

Operating system:
• Manages the available hardware (CPU, memory...) and software (editors, development tools...) resources.
• Provides interface for using them by multiple users.

Shell: command-line interpreter to interface the OS.
The Unix OS

Unix: multi-user, multi-process OS; open source.

History:
- 1969 – Initial version by Thompson & Ritchie at Bell Labs (assembly and later B).
- 70’s – Rewritten in C.
- 80’s – System-V (AT&T) and BSD (Berkeley) versions.
- 90’s – Linux by Linus Torvalds.

For basic introduction and commands see web-page.
Programming Project

Goals:

• Build software system.
• Memory management.
• Fit spec & understand complex requirements.
• Basic file & disc handling.
• Receive & interface with external code.
• Testing and debug purpose functions.
Project story

WindowsME had a serious weakness in password authentication method - trust “one-way” function that is actually easy to invert.

Breaking using rainbow-table
• Preprocessing all possible passwords, and save it efficiently.
• Disc-space vs. Access-time tradeoff.
Assignments

• Ex1 (5%) : Break RSA on 32bit keys. Flow-control, smart use of variables.
• Ex2 (5%) : Legal password generation. String manipulation, simplest memory management.
• Ex3(10%): Implementing hash-table Lists, correct memory management.
• Project (50%): Passwords authentication, and its retrieving using rainbow tables. Large system, accessing to disc, files. Huge runs (debug with small runs)
From source code to executable

Three main steps:

- **Preprocessing**: First pass on the file that substitutes human-friendly text with machine-friendly representation
- **Compilation**: Generation of object code
- **Linking**: Combining several object code files into a single executable set of instructions
Example 1

```c
#include <stdio.h>
int main(void) {
    printf("Hello, world!\n");
    return 0;
}
```

Source -> preprocessing -> compiling -> linking -> executable

```
hello.c       "gcc –c hello.c"  "gcc hello.o"  hello.exe
```