Programming in C

Based on the Original Slides from
Politehnica International- Computer Engineering
Lecture Slides
Lecture 1: Outline

• Introductions
• Course Logistics and Syllabus
  – Learning Objectives
  – Textbook
  – Grading
• Some Fundamentals
• Compiling and running your first C program
Learning Objectives

• First course in Computer Programming Using C
  – No previous knowledge is assumed!
• By the end of the course, students will:
  – Understand fundamental concepts of computer
    Procedural programming languages
  – Design algorithms to solve (simple) problems
  – Use the C programming language
Textbook

- Kochen, Programming in C, Third Edition

  - Is considered “THE” book on C: coauthor belongs to the creators of the C programming language
  - The book is not an introductory programming manual; it assumes some familiarity with basic programming concepts

- C Programming Notes by Steve Summit

  http://www.eskimo.com/~scs/cclass/notes/top.html
Policies and Grading

- Lectures: can be interactive, with questions and interactive problem solving
- Some quizzes require in-class programming
- Assignments should be solved individually
- Attendance is recorded (at a random point) during the class time.
- Grading:
  - Assignments: 20%
  - Quizes: 15%
  - Exam I: 20%
  - Exam II: 20%
  - Final: 20%
  - Attendance: 5%
Course Topics

- Some Fundamentals
- Compiling and Running your First C Program
- Variables, Data Types, and Arithmetic Expressions
- Program Looping
- Making Decisions
- Working with Arrays
- Working with Functions
- Working with Structures
- Character Strings
- Pointers
- Operations on Bits
- The Preprocessor
- More on Data Types
- Working with Larger Programs
- Input and Output Operations in C
- Miscellaneous and Advanced Features
Fundamentals

• Classical model for computing machines
• Programming
• Programming languages
• Compiling
• Operating system
Model of a computing machine

• Computing machine (Computer): “a machine that *stores and manipulates information* under the control of a *changeable program* that is *stored in its memory*.”

  • Pocket calculator: not a computer! Manipulates information, but is built to do a specific task (no changeable stored program)

• This model is named the “von Neumann architecture” (John von Neumann – 1945; EDVAC - Electronic Discrete Variable Automatic Computer – the first stored-program computer)

• Stored-program concept: earlier ideas in theoretical articles of: Alan Turing (1936), Konrad Zuse (1936)

• Optional reading: history of computing – IEEE Computing Society – timeline of occasions in computing history
  http://www.computer.org/portal/cms_docs_ieeeecs/ieee/cs/about/history/timeline.pdf
The von Neumann architecture

- **Central Processing Unit (CPU):** the “brain” of the machine.
  - Carries out all basic operations of the computer
  - Examples of basic operation: adding two numbers, testing to see if two numbers are equal.
- **Main memory (called RAM for Random Access Memory):** stores programs and data
  - Fast but volatile
- **Secondary memory:** provides permanent storage
- **Human-computer interaction:** through input and output devices.
  - keyboard, mouse, monitor
  - Information from input devices is processed by the CPU and may be sent to the main or secondary memory. When information needs to be displayed, the CPU sends it to the output device(s).
How it works

• How does a computer execute a program? (example programs: a computer game, a word processor, etc)
• the instructions that comprise the program are copied from the permanent secondary memory into the main memory
• After the instructions are loaded, the CPU starts executing the program.
• For each instruction, the instruction is retrieved from memory, decoded to figure out what it represents, and the appropriate action carried out. (the *fetch, decode, execute cycle*)
• Then the next instruction is fetched, decoded and executed.
Suppose we want the computer to add two numbers. The instructions that the CPU carries out might be:
- load the number from memory location 1006 into the CPU
- load the number from memory location 2345 into the CPU
- add the two numbers in the CPU
- store the result into memory location 2981

The processors instruction set: all basic operations that can be carried out by a certain type of processor
- the instructions and operands are represented in binary notation (sequences of 0s and 1s).
  - Why binary? Because computer hardware relies on electric/electronic circuits that have/can switch between 2 states
  - bit (binary digit)
  - Byte: 8 bits

The program carried out by the CPU, on a certain processor type, could look like:
- 1010 1111
- 0011 0111
- 0111 0110
- ...

This way had to be programmed the first computers!
- The job of the first programmers was to code directly in machine language and to enter their programs using switches
Higher level languages

- **Assembly language**
  - First step from machine language
  - Uses symbolic names for operations and to refer to memory locations
  - Translation of assembly language into machine language: in the beginning done manually, later done by a special computer program – the *assembler*
  - Low-level language:
    - programmer must learn the instruction set of the particular processor
    - Program must be rewritten in order to run on a different processor type – program is not *portable*

- **High level languages**
  - Writing portable programs, using more abstract instructions
  - A high level instruction (statement) is translated into many machine instructions
  - Translation of high level language into machine instructions: done by special computer programs – *compilers or interpreters*
  - First high-level language: FORTRAN
Compiler: analyzes program and translates it into machine language. Executable program: can be run independently from compiler as many times => fast execution.

Interpreter: analyzes and executes program statements at the same time. Execution is slower. Easier to debug program.
Operating Systems

• Operating system: a program that controls the entire operation of a computer system:
  – Handles all input and output (I/O) operations that are performed on a computer
  – manages the computer system’s resources
  – handles the execution of programs (including multitasking or multiuser facilities)

• Most famous OS families:
  – Windows
  – Unix
The C Programming Language

- Developed by Dennis Ritchie at AT&T Bell Laboratories in the early 1970s
- Growth of C tightly coupled with growth of Unix: Unix was written mostly in C
- Success of PCs: need of porting C on MS-DOS
- Many providers of C compilers for many different platforms => need for standardization of the C language
- 1990: ANSI C (American National Standards Institute)
The first C program

uses standard library input and output functions (printf)

begin of program statements

end of program

main: a special name that indicates where the program must begin execution. It is a special function.

first statement: calls a routine named printf, with argument the string of characters “Programming is fun \n”

last statement: finishes execution of main and return to the system a status value of 0 (conventional value for OK)

#include <stdio.h>

int main (void)
{
    printf ("Programming is fun.\n");
    return 0;
}
The format in C

• Statements are terminated with semicolons
• Indentation is nice to be used for increased readability.
• Free format: white spaces and indentation is ignored by compiler
• C is case sensitive – pay attention to lower and upper case letters when typing!
  – All C keywords and standard functions are lower case
  – Typing INT, Int, etc instead of int is a compiler error
• Strings are placed in double quotes
• New line is represented by \n (Escape sequence)
Compiling and running C programs

Editor

Source code
file.c

Compiler

Object code
file.obj

Linker

Executable code
file.exe

IDE (Integrated Development Environment)

Libraries
C Compilers and IDE’s

• One can:
  – use a text editor to edit source code, and then use independent command-line compilers and linkers
  – use an IDE: everything together + facilities to debug, develop and organize large projects

• There are several C compilers and IDE’s that support various C compilers

• We will use Visual C++ 2010 and the “cl” compiler.

  cl prog.c ➔ porg.exe
Debugging program errors

- **Editor**
  - Source code file.c

- **Compiler**
  - Object code file.obj

- **Linker**
  - Executable code file.exe

- **Libraries**

**Syntactic Errors**

**Semantic Errors**
Syntax and Semantics

- Syntax errors: violation of programming language rules (grammar)
  - "Me speak English good."
  - Use valid C symbols in wrong places
  - Detected by the compiler
- Semantics errors: errors in meaning:
  - "This sentence is excellent Italian."
  - Programs are syntactically correct but don’t produce the expected output
  - User observes output of running program
Second program

```c
#include <stdio.h>
int main (void)
{
    printf ("Programming is fun.\n");
    printf ("And programming in C is even more fun.\n");
    return 0;
}
```
Displaying multiple lines of text

```c
#include <stdio.h>
int main (void)
{
    printf ("Testing...
...1
...2
....3
");
    return 0;
}
```

Output:

```
Testing...
..1
...2
....3
```

It is not necessary to make a separate call to printf for each line of output!
Variables

• Programs can use symbolic names for storing computation data and results

• Variable: a symbolic name for a memory location
  – programmer doesn’t have to worry about specifying (or even knowing) the value of the location’s address

• In C, variables have to be declared before they are used
Using and Displaying Variables

```c
#include <stdio.h>
int main (void)
{
    int sum;
    sum = 50 + 25;
    printf ("The sum of 50 and 25 is %i\n", sum);
    return 0;
}
```

Variable sum **declared** of type int

Variable sum **assigned** expression 50+25

Value of variable sum is **printed** in place of %i

The printf routine call has now 2 arguments: first argument a string containing also a format specifier (%i), that holds place for an integer value to be inserted here
Displaying multiple values

```c
#include <stdio.h>
int main (void)
{
    int value1, value2, sum;
    value1 = 50;
    value2 = 25;
    sum = value1 + value2;
    printf ("The sum of %i and %i is %i\n", value1, value2, sum);
    return 0;
}
```

The format string must contain as many placeholders as expressions to be printed
Using comments in a program

• Comment statements are used in a program to document it and to enhance its readability.
• Useful for human readers of the program – compiler ignores comments
• Ways to insert comments in C:
  – When comments span several lines: start marked with /*, end marked with */
  – Comments at the end of a line: start marked with //
Using comments in a program

/* This program adds two integer values and displays the results */

#include <stdio.h>
int main (void)
{
    // Declare variables
    int value1, value2, sum;
    // Assign values and calculate their sum
    value1 = 50;
    value2 = 25;
    sum = value1 + value2;
    // Display the result
    printf ("The sum of %i and %i is %i\n", value1, value2, sum);
    return 0;
}