Lecture 2: Overview

- Introduction to Programming in C
  - History of C
  - Introduction to C
- Our first C Program
  - Example HelloWorld.c
  - Structure of a C program
  - `printf()` function
  - Program compilation and execution
  - String constants
Introduction to Programming

- Categories of programming languages
  - Machine languages (stream of 1’s and 0’s)
  - Assembly languages (low-level CPU instructions)
  - High-level languages (high-level instructions)

- Translation of high-level languages
  - Interpreter (translation for each instruction)
  - Compiler (translation once for all code)
  - Hybrid (combination of the above)

- Types of programming languages
  - Functional (e.g. Lisp)
  - Structured (e.g. Pascal, C, Ada)
  - Object-oriented (e.g. C++, Java, Python)
Introduction to Programming

- Categories of programming languages
  - Machine languages (stream of 1’s and 0’s)
  - Assembly languages (low-level CPU instructions)
  - High-level languages (high-level instructions)

- Translation of high-level languages
  - Interpreter (translation for each instruction)
  - Compiler (translation once for all code)
  - Hybrid (combination of the above)

- Types of programming languages
  - Functional (e.g. Lisp)
  - Structured (e.g. Pascal, C, Ada)
  - Object-oriented (e.g. C++, Java, Python)
Introduction to Programming

- Categories of programming languages
  - Machine languages (stream of 1’s and 0’s)
  - Assembly languages (low-level CPU instructions)
  - High-level languages (high-level instructions)
- Translation of high-level languages
  - Interpreter (translation for each instruction)
  - Compiler (translation once for all code)
  - Hybrid (combination of the above)
- Types of programming languages
  - Functional (e.g. Lisp)
  - Structured (e.g. Pascal, C, Ada)
  - Object-oriented (e.g. C++, Java, Python)
Introduction to Programming

- Categories of programming languages
  - Machine languages (stream of 1’s and 0’s)
  - Assembly languages (low-level CPU instructions)
  - High-level languages (high-level instructions)

- Translation of high-level languages
  - Interpreter (translation for each instruction)
  - Compiler (translation once for all code)
  - Hybrid (combination of the above)

- Types of programming languages
  - Functional (e.g. Lisp)
  - Structured (e.g. Pascal, C, Ada)
  - Object-oriented (e.g. C++, Java, Python)
History of C

• Evolved from BCPL and B
  • in the 60’s and 70’s
• Created in 1972 by Dennis Ritchie (Bell Labs)
  • development language of UNIX operating system
• “Traditional” C
  • 1978, "The C Programming Language”,
    by Brian W. Kernighan, Dennis M. Ritchie
  • ported to most platforms
• ANSI C
  • standardized in 1989 by ANSI and OSI
  • standard updated in 1999
Introduction to C

- What is C?
  - Programming language
    - high-level
    - structured
    - compiled
  - Standard library
    - collection of existing functions
- Why C?
  - de-facto standard in software development
  - code is portable to many different platforms
  - easy transition to object-oriented programming
    - C++ / Java
  - freely available for most platforms
Our first C Program

- Program example: `HelloWorld.c`

```c
/* HelloWorld.c: our first C program */
/* */
/* author: Rainer Doemer */
/* */
/* */
/* modifications: */
/* */
/* 09/28/04 RD initial version */

#include <stdio.h>

/* main function */

int main(void)
{
    printf("Hello World!\n");
    return 0;
}

/* EOF */
```
Our first C Program

- Program comments
  - start with /* and end with */
  - are ignored by the compiler
  - should be used to
    - document the program code
    - structure the program code
    - enhance the readability
- `#include` preprocessor directive
  - inserts a header file into the code
- standard header file `<stdio.h>
  - part of the C standard library
  - contains declarations of standard types and functions
    for data input and output (e.g. function `printf()`)

```c
/* HelloWorld.c: our first C program */
/* author: Rainer Doemer */
/* modifications: */
/* 09/28/04 RD initial version */
#include <stdio.h>
/* main function */
int main(void)
{
    printf("Hello World!\n");
    return 0;
}
/* EOF */
```
Our first C Program

- **int main(void)**
  - main function of the C program
  - the program execution starts (and ends) here
  - `main` must return an integer (`int`) value to the operating system at the end of its execution
    - return value of 0 indicates successful completion
    - return value greater than 0 usually indicates an error condition
- function body
  - block of code (definitions and statements)
    - starts with an opening brace (`{`)
    - ends with a closing brace (`}`)
  - `printf()` function
    - formatted output (to stdout)
  - `return` statement
    - ends a function and returns its argument as result

```c
... /* main function */
int main(void)
{
    printf("Hello World!\n");
    return 0;
}
/* EOF */
```
Our first C Program

- Program compilation
  - compiler translates the code into an executable program
  - `gcc HelloWorld.c`
  - compiler reads file `HelloWorld.c` and creates file `a.out`
  - options may be specified to direct the compilation
    - `-o HelloWorld` specifies output file name
- Program execution
  - use the generated executable as command
    - `HelloWorld`
  - the operating system loads the program (loader), then executes its instructions (program execution), and finally resumes when the program has terminated
Our first C Program

- Example session: HelloWorld.c

```
east% mkdir HelloWorld
east% cd HelloWorld
east% ls
east% vi HelloWorld.c
east% ls
HelloWorld.c
east% ls -l
-rw-r--r-- 1 doemer faculty 263 Sep 28 22:11 HelloWorld.c
east% gcc HelloWorld.c
east% ls -l
-rw-r--r-- 1 doemer faculty 263 Sep 28 22:11 HelloWorld.c
-rwxr-xr-x 1 doemer faculty 6352 Sep 28 22:12 a.out*
east% a.out
Hello World!
east% gcc -Wall -ansi HelloWorld.c -o HelloWorld
east% ls -l
-rwxr-xr-x 1 doemer faculty 6356 Sep 28 22:17 HelloWorld*
-rw-r--r-- 1 doemer faculty 263 Sep 28 22:17 HelloWorld.c
-rw-r--r-- 1 doemer faculty 263 Sep 28 22:17 HelloWorld.c
-rw-r-xr-x 1 doemer faculty 6352 Sep 28 22:12 a.out*
east% HelloWorld
Hello World!
```
Our first C Program

- Character string constants: “Strings”
  - start and end with a double quote character ("")
  - may not extend over a single line
  - subsequent string constants are combined
  - text formatting using escape sequences
    - \n  new line
    - \t horizontal tab
    - \r carriage return
    - \b back space
    - \a alert / bell
    - \\ backslash character
    - \" double quote character
- Experiments with the HelloWorld program...
Performing Computation
Program Structure

- General Program Structure
  - Input
    - read input data
  - Computation
    - compute output data from input data
  - Output
    - write output data
- Examples
  - Calculator
    - Enter numbers, compute function, output result
  - Word processor
    - Type, format, print text
  - Database application
    - Enter data, process data, present data
  - etc.
C Program Structure

- Initialization section
  - Definition of variables (storage elements)
    - Name, type, and initial value
- Input section
  - read values from input devices into variables
    - standard input functions
- Computation section
  - perform the necessary computation on variables
    - assignment statements
- Output section
  - write results from variables to output devices
    - standard output functions
- Exit section
  - clean up and exit
Our second C Program

- **Program example: Addition.c (part 1/2)**

```c
/* Addition.c: adding two integer numbers */
/* */
/* author: Rainer Doemer */
/* */
/* */
/* modifications: */
/* */
/* 09/30/04 RD initial version */

#include <stdio.h>

/* main function */

int main(void)
{
    /* variable definitions */
    int i1 = 0;    /* first integer */
    int i2 = 0;    /* second integer */
    int sum;      /* result */

    ...
```
Our second C Program

- Program example: `Addition.c` (part 2/2)

```c
... /* input section */
    printf("Please enter an integer: ");
    scanf("%d", &i1);
    printf("Please enter another integer: ");
    scanf("%d", &i2);

    /* computation section */
    sum = i1 + i2;

    /* output section */
    printf("The sum of %d and %d is %d.\n", i1, i2, sum);

    /* exit */
    return 0;
} /* end of main */
/* EOF */
```
Our second C Program

• Variable definition and initialization

```c
/* variable definitions */
int i1 = 0; /* first integer */
int i2 = 0; /* second integer */
int sum;    /* result */
```

• Variable type: int
  • integer type, stores whole numbers (e.g. -5, 0, 42)
  • many other types exist (float, double, char, ...)

• Variable name: i1, i2, sum
  • valid identifier, i.e. name composed of letters, digits
  • variable name should be descriptive

• Initializer: = 0
  • specifies the initial value of the variable
  • optional (if omitted, initial value is undefined)
Our second C Program

- Data input using `scanf()` function

```c
/* input section */
printf("Please enter an integer:        ");
scanf("%d", &i1);
```

- part of standard I/O library
  - declared in header file `stdio.h`
- reads data from the standard input stream `stdin`
  - `stdin` usually means the keyboard
- converts input data according to format string
  - "%d" indicates that a decimal integer value is expected
- stores result in specified location
  - `&i1` indicates to store at the _address of_ variable `i1`
Our second C Program

- Computation using assignment statements

```c
/* computation section */
sum = i1 + i2;
```

- Operator `+` specifies addition
  - left and right arguments are added
  - result is the sum of the two arguments
- Operator `=` specifies an assignment
  - value of the right-hand side (`i1 + i2`) is assigned to the left-hand side (`sum`)
  - left-hand side is usually a variable
  - right-hand side is a simple or complex expression
- May other operators exist
  - For example, `-`, `*`, `/`, `%`, `<`, `>`, `==`, `^`, `&`, `|`, ...

Our second C Program

- Data output using `printf()` function
  ```c
  /* output section */
  printf("The sum of %d and %d is %d.\n", i1, i2, sum);
  ```
- part of standard I/O library
  - declared in header file `stdio.h`
- writes data to the standard output stream `stdout`
  - `stdout` usually means the monitor
- converts output data according to format string
  - standard text is copied verbatim to the output
  - "\%d" is replaced with a decimal integer value
- takes values from specified arguments
  - `i1` indicates to use the value of the variable `i1`
Our second C Program

- Example session: Addition.c

```
% vi Addition.c
% ls -l
-rw------- 1 doemer faculty  702 Sep 30 14:17 Addition.c
% gcc -Wall -ansi Addition.c -o Addition
% ls -l
-rwx------- 1 doemer faculty  6628 Sep 30 16:44 Addition*
-rw------- 1 doemer faculty  702 Sep 30 14:17 Addition.c
% Addition
Please enter an integer: 27
Please enter another integer: 15
The sum of 27 and 15 is 42.
% Addition
Please enter an integer: 123
Please enter another integer: -456
The sum of 123 and -456 is -333.
```