Introduction

- derived from the slides of the book “C++ For Everyone” by Cay Horstmann
- slides by Evan Gallagher

Goals

- to learn about machine languages
  and higher-level programming languages
- to become familiar with your compiler
- to compile and run your first C program
- to recognize compile-time and run-time errors
- to describe an algorithm with pseudocode
- to understand the activity of programming

What Is Programming?

- program: tells a computer, in minute detail,
  the sequence of steps that are needed to fulfill a task.

- programming: act of designing and implementing
  computer programs

Machine Code

- programs stored as machine instructions
- in a code that depends on the processor type

- example sequence of machine instructions:

  1. move contents of memory location 40000 into CPU
  2. if that value is > 100, continue with instruction that is stored in memory location 11280

Machine Code

- machine instructions encoded as numbers
- on a PC, the instructions from the previous slide are encoded as::

  161 40000 45 100 127 11280
- on a different processor, the encoding would be different

High-Level Languages

- high-level languages are independent of processor type and hardware
- source code: program in high-level language

- compiler: a program that translates a source code into machine code
- **for a particular processor**

Portability

- compiler-generated machine code
  is different between processors
- programmer need not worry about these differences

Evolution of C

- Dennis Ritchie (1972)
- ANSI Standard C (1989, 1999, 2011)

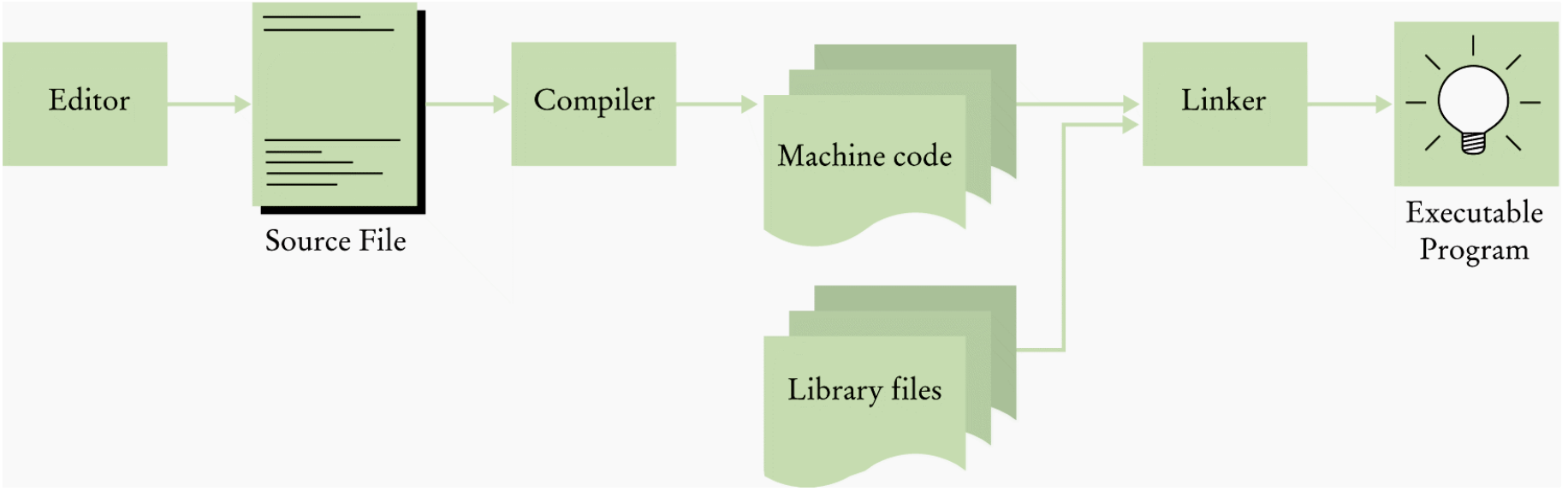
- Bjarne Stroustrup: C++ (1985)
- adds features (object oriented)

Development Environment

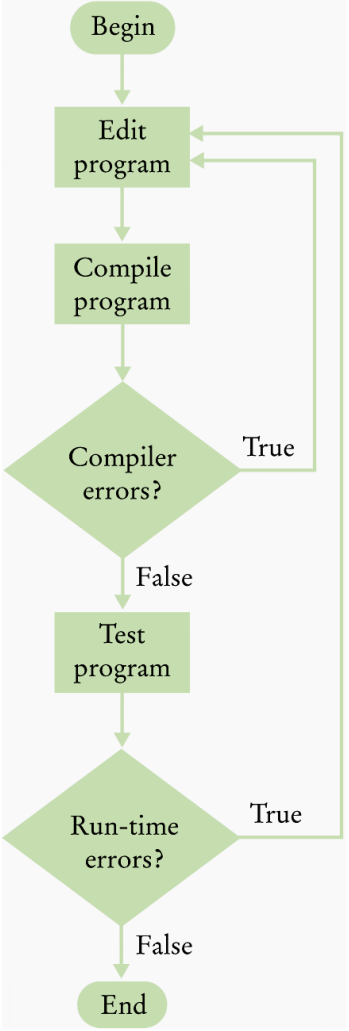
- write code: editor
- build executable: compiler
- run executable

- integrated development environment (IDE)

Development Stages



Development Workflow



Filename Conventions

- C source code extension:

  `.c`

- C++ source code extension:

  `.cpp`

  `.cc`

  `.cxx`

First Program

- Hello, world!
- write the words “Hello, world!” on the screen

Hello, world!

```c
#include <stdio.h>

int main()
{
   printf("Hello, world!\n");
   return 0;
}
```

Hello, world!

- the book is based on C++
- we will not use this

```
1
2
3
4
5
6
7
8
9
```

```cpp
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello, world!" << endl;
   return 0;
}
```

Headers

```
1
2
3
4
5
6
7
```

```c
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

- programs include “headers” for required services
- line 1: include “standard input and output” service

Starting Point

```
1
2
3
4
5
6
7
```

```c
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

- line 3: every program has a `main` function
- where the program starts

Braces

```
1
2
3
4
5
6
7
```

```c
#include <stdio.h>

int main()
{
   printf("Hello, world!\n");
   return 0;
}
```

- line 4 and 7: statements of a function are enclosed in braces

Semicolons

```
1
2
3
4
5
6
7
```

```c
#include <stdio.h>

int main()
{
   printf("Hello, world!\n");
   return 0;
}
```

- line 5 and 6: statements end with a semicolon

Program Result

```
1
2
3
4
5
6
7
```

```c
#include <stdio.h>

int main()
{
   printf("Hello, world!\n");
   return 0;
}
```

- line 6: `main` function returns an integer
- 0: success

Printing Messages

```
1
2
3
4
5
6
7
```

```c
#include <stdio.h>

int main()
{
   printf("Hello, world!\n");
   return 0;
}
```

- line 5: to show output on the screen,
  use a function called `printf`

String Literals

- `"Hello, world!\n"` is a string literal
- enclosed in double quotes

- `\n` denotes end of line
- move cursor to the next screen row

Compiling and Linking

- compile:

```
gcc -c -std=c99 hello.c -o hello.o
```

- link:

```
gcc hello.o -o hello
```

Combining Stages

- compile and link:

```
gcc -std=c99 hello.c -o hello
```

Errors

- errors: no executable gets built

- warnings: executable gets built
- but possibly won’t work as expected
- don’t ignore warnings

Error Switches

- show all warnings:

```
gcc -std=c99 -Wall hello.c -o hello
```

- treat warnings as errors:

```
gcc -std=c99 -Werror hello.c -o hello
```

- this is the mode we will use

Omitting Semicolons

- common error: forgetting a semicolon

```
1
2
3
4
5
6
7
```

```c
#include <stdio.h>

int main()
{
   printf("Hello, world!\n")
   return 0;
}
```

- line 5

Omitting Semicolons

- without the semicolon, you actually wrote:

```
1
2
3
4
5
6
```

```c
#include <stdio.h>

int main()
{
   printf("Hello, world!\n") return 0;
}
```

- syntax error: not conforming to the rules of the PL
- caught at compile time

Undefined Names

- using undefined names

```
1
2
3
4
5
6
7
```

```c
#include <stdio.h>

int main()
{
    print("Hello, world!\n");
    return 0;
}
```

- line 5: `print` instead of `printf`
- warning at compile time, error at link time

Multiple main Functions

- every C program must have one and only one `main` function

Case Sensitivity

- C is case sensitive

```
1
2
3
4
5
6
7
```

```c
#include <stdio.h>

int Main()
{
    printf("Hello, world!\n");
    return 0;
}
```

- line 3: `Main` instead of `main`

Case Sensitivity

- this will compile but not link
- linker cannot find the `main` function
- because you did not define one
- `Main` is fine as a name but it is not the same as `main`
- and there has to be one main somewhere

Many Errors

- compiler will not stop compiling at first error
- it will most likely list lots and lots of errors
  that are caused by the first one

- fix only those error messages that make sense to you
- starting with the first one, and recompile

Code Layout

- C has free-form layout
- this will work:

```
#include <stdio.h>

int main(){printf("Hello, world!\n");return 0;}
```

- but is not readable
- a good program is readable

Algorithm

- algorithm: step by step description of a solution
- like a recipe

- unambiguous: precise instructions for each step
- terminating: must not run forever

Pseudocode

- pseudocode: informal description of an algorithm

- not a PL source code
- but easily translated into one

Pseudocode Example

- you have the choice of buying two cars
- one is more fuel efficient but also more expensive
- you know the price and fuel efficiency
  (in miles per gallon, mpg) of both cars

Pseudocode Example

- you plan to keep the car for 10 years
- assume a price of gas is $4 per gallon and usage of 15,000 miles per year
- you will pay cash for the car
- which car is the better deal?

Pseudocode Example - Step 1

- step 1: determine inputs and outputs
- inputs:
  - purchase price and fuel efficiency of first car
  - purchase price and fuel efficiency of second car
- output: the better car to buy

Pseudocode Example - Step 2

- step 2: break down the problem into smaller tasks

- total cost for a car: purchase price + operating cost
- assuming constant usage and gas price for 10 years
- operating cost: 10 x annual fuel cost
- annual fuel cost: price per gallon x annual fuel consumed
- annual fuel consumed: annual miles driven / fuel efficiency

Pseudocode Example - Step 3

- step 3: describe each subtask in pseudocode
- arrange steps so that any intermediate values are computed before they are needed

Pseudocode Example - Step 3

- for each car, compute the total cost:

```
annual fuel consumed = annual miles driven / fuel efficiency
annual fuel cost = price per gallon x annual fuel consumed
operating cost = 10 x annual fuel cost
total cost = purchase price + operating cost
```

- compare the costs:

```
If total cost1 < total cost2
    Choose car1
Else
    Choose car2
```

Pseudocode Example - Step 4

- step 4: test your pseudocode by working a problem

- use these sample values:
  Car 1: $25,000, 50 miles/gallon
  Car 2: $20,000, 30 miles/gallon

Pseudocode Example - Step 4

- Car 1:

```
annual fuel consumed = 15000 / 50 = 300
annual fuel cost = 4 x 300 = 1200
operating cost = 10 x 1200 = 12000
total cost = 25000 + 12000 = 37000
```

- Car 2:

```
total cost = 40000
```

Pseudocode Example - Step 4

- if total cost1 < total cost2 …
- algorithm says: choose Car 1