Getting Started in C

ITCS 2116: C Programming
College of Computing and Informatics
Department of Computer Science

UNIVERSITY OF NORTH CAROLINA
CHARLOTTE

Outline

- C Overview
- Software Tools
- Course Goals
- Programming Languages
- Common Platform
- Sample Program

Why C?

- Developed to build Unix operating system
- Main design considerations:
  - Compiler size: needed to run on PDP-11 with 24KB of memory (Algol60 was too big to fit)
  - Code size: needed to implement the whole OS and applications with little memory
  - Performance
  - Portability
- Little consideration (if any) to the following:
  - Security, robustness, maintainability
  - Legacy Code

Why C? (cont’d)

- Simple to write compiler
  - Programming embedded systems, often only have a C compiler
- Performance
  - Typically 50x faster than interpreted Java
- Smaller, simpler, lots of experience
- One of the most popular programming languages
  - For the latest numbers, see https://www.tiobe.com/tiobe-index/

What’s Your Priority?

Priority	Language Choices
Speed of execution, minimum memory “footprint”	Assembly, C
Safer, easier to develop large (hundreds of files) programs	Java, C++
Easier / faster to code, higher level operations, richer libraries	Python, Ruby, PHP, Perl
Integrate with the web	Web application frameworks, Javascript

C Strengths

- It’s a **procedural** language (like many others
- It’s **efficient** (binary code size, execution speed)
- **Simple**, **clean** language design
- C99 is a **international standard**
- It has a decent **standard library** of useful functions

Examples of C or C++

- **Linux**: Assembly, C
- **MS Windows**: Assembly, C, C++
- **Firefox Web Browser**: C++, Javascript
- **Gnu Compiler** (GCC): C
- **MySQL**: C, C++
- **Embedded Systems** (cars, appliances, etc.)
- **High performance** (science/engineering) applications

C Weaknesses

- **Little** consideration for **security** or **safety**
- **Less modular** than Java and other OO languages (but C++ fixes that)
- **More programming effort** required than PHP/Python/Perl/Ruby and other scripting languages
- **Not** usually written in C or C++: **web apps, business apps, GUIs, simple utility programs**

Software Tools

- Help produce programs…
  - Quickly
  - Of high quality
  - More reliably
  - In large teams of programmers
- Examples of tools
  - Compilers, code formatters / indenters, debuggers, test generators, performance profilers, version control management, dependency checking, documentation generation, static analysis, …
- Often these are bundled in an IDE
  - Eclipse, Visual Studio, …

Some Standard Goals

- Understand syntax and semantics of C and how to use
- Be able to write small- to medium-sized C programs
- Understand differences between compiling and interpreting
- Know how to avoid, find, and fix programming bugs in C
- Know how to dynamically allocate/free memory
- Know how to use header files and the C preprocessor

Some Standard Goals (cont’d)

- Be familiar with and know how to use standard library functions
- Use command-line tools to design, compile, document, debug, improve, and maintain programs
- Know how to automate dependence checking / building an executable / common programming tasks
- Know how to use common tools to write programs as part of a team

Other Goals

- Will this course make me…
  - ✓ a better programmer?
  - ✓ a better computer scientist?
  - ✓ more marketable?
  - ? wealthy, successful, famous?
  - ? a better person?

Types of Programming Languages

- **Declarative**: focus on what the computer should do
  - *Functional:* Scheme, Haskell
  - *Dataflow*
  - *Logic-* or *constraint-based: Prolog*
  - *Markup languages:* HTML, CSS, subset of SQL
- **Imperative:** focus on how the computer should do something
  - *Procedural* : **C**
  - *Object-oriented* : **Java**

Procedural vs. Object-Oriented

- **Procedural**: programming as procedures that modify variables
  - Emphasis on actions that must take place
  - Analogy: following a recipe
- **Object-Oriented**: programming as objects that interact (each with internal state, and methods to manage that state)
  - Emphasis on the state of objects
  - Analogy: operating a car

Getting Started….

Common Platform for This Course

- Different platforms have different conventions for end of line, end of file, tabs, compiler output, …
- Solution (for this class): **compile and run** all programs consistently **on one platform**
- Our common platform:
  - Replit (repl.it)

Your Choices

- Use **ReplIt (**repl.it)**,** a web-based virtual computing environment (https://replit.com/)
- Use a CCI Lab Computer
- Use Mac OS X (Xcode + developer tools)
- Use MS Windows + **`cygwin`** or Visual Studio
- Use Linux on your PC (dual boot or virtualized)

Common Platform Questions

- If you want to develop locally, that’s fine, but you must ensure that it works on the Common Platform
  - You should always test on the Common Platform before submitting
  - The Instructional Assistants will use the common platform to grade your work
  - **No, really, you should test on the Common Platform**

Common Platform Questions

- There are differences between the C compilers for different architectures that may cause your program (that runs locally) to fail on the Common Platform
- C is not architecture neutral!

Your First C Program

```
#include <stdio.h>

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

File with library function declarations

Entry point of the program, with no arguments

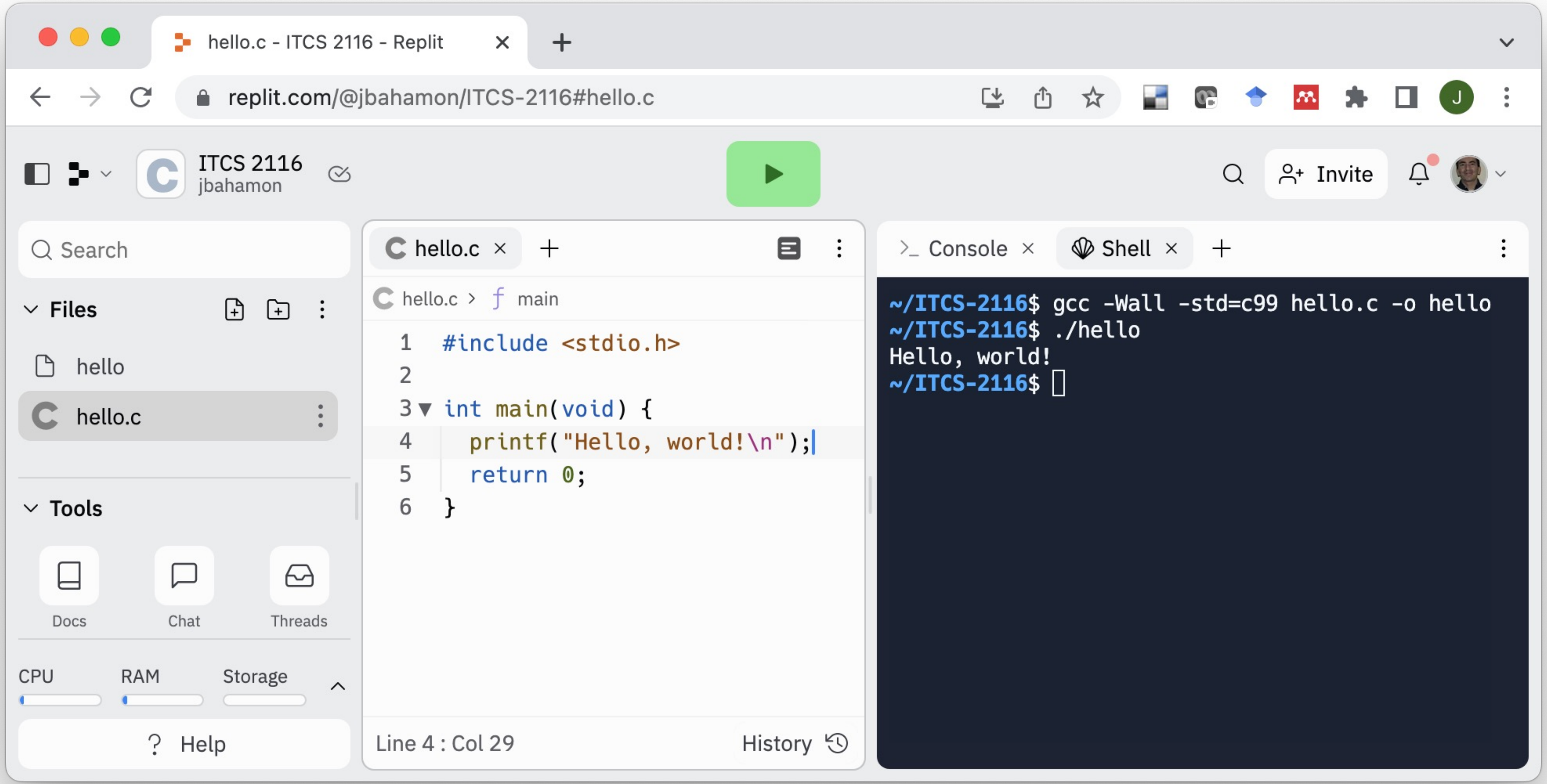
Standard library function, with message argument

Exit program and indicate successful completion

Command to compile program code into an executable

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

Compiling and Running the Program



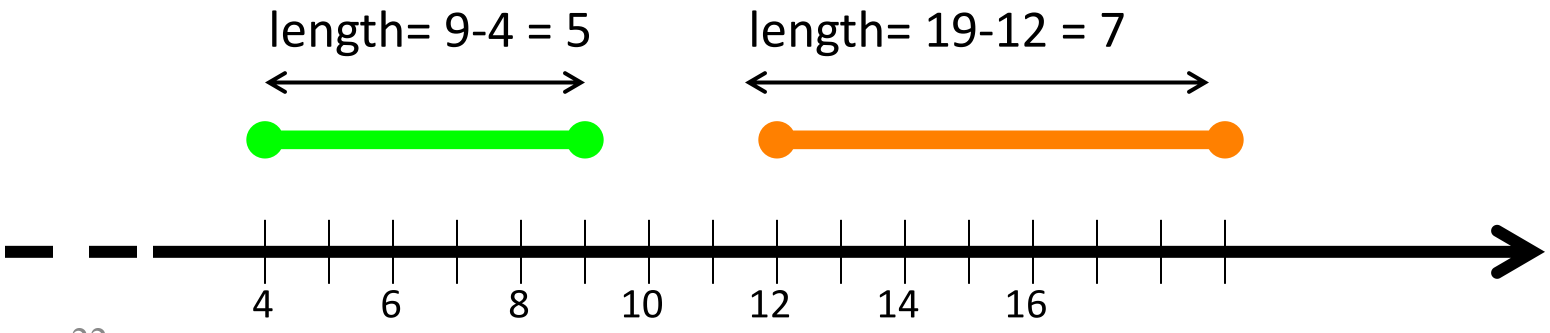
```c
#include <stdio.h>

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

```
~/ITCS-2116$ gcc -Wall -std=c99 hello.c -o hello
~/ITCS-2116$ ./hello
Hello, world!
~/ITCS-2116$
```

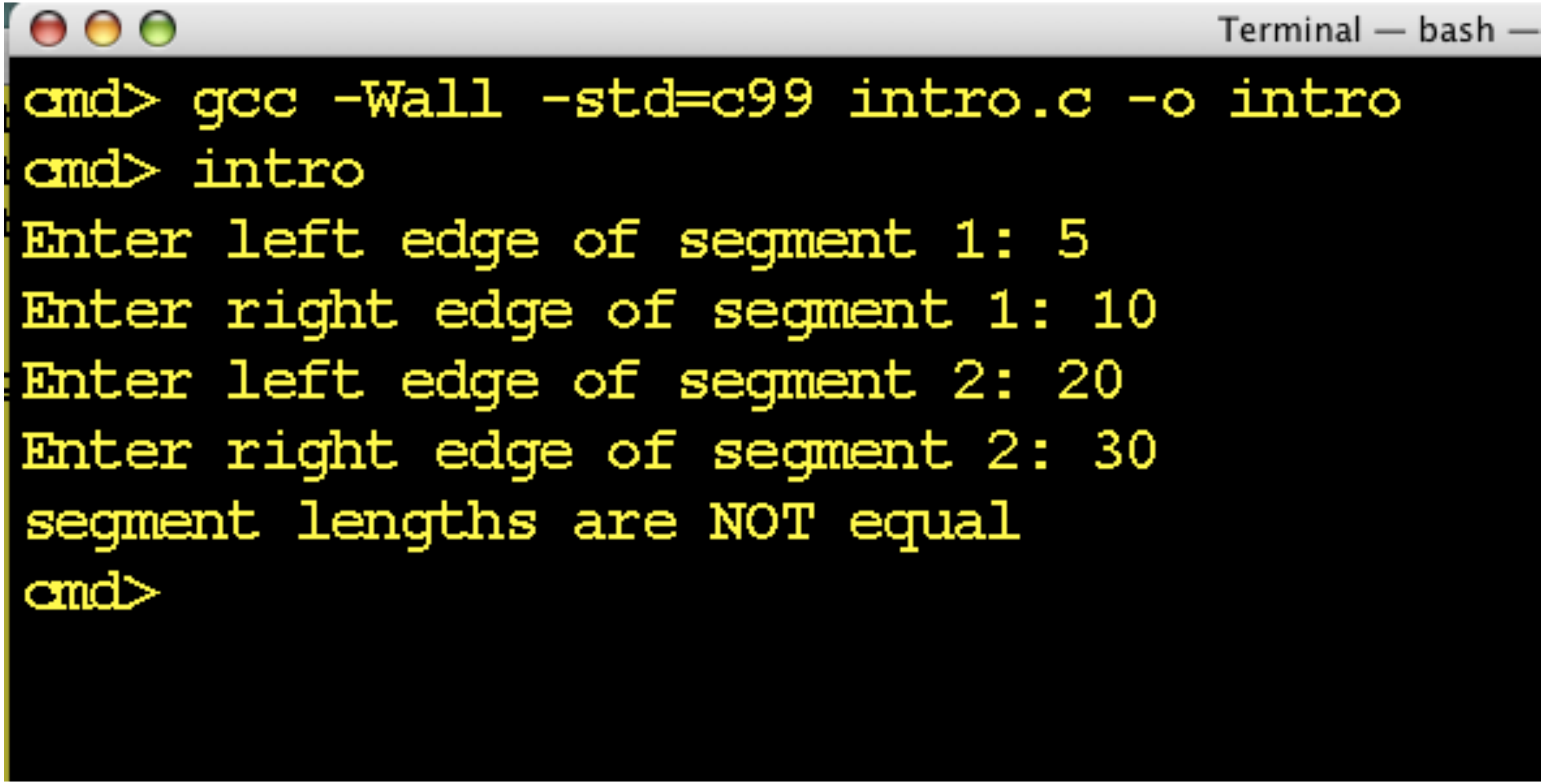
A Sample Program (For Illustration)

Specification:

1. Two line segments are created
2. The user is asked to enter the left and right edges of the two line segments, as integer values
3. The length of each segment is computed as (*right edge – left edge*)
4. The two lengths are compared to determine if they are the same, and a message is displayed



Compiling and Running the Program



```
cmd> gcc -Wall -std=c99 intro.c -o intro
cmd> intro
Enter left edge of segment 1: 5
Enter right edge of segment 1: 10
Enter left edge of segment 2: 20
Enter right edge of segment 2: 30
segment lengths are NOT equal
cmd>
```

Sample *C* Program (part 1)

- The following slides show sample program code to implement a solution to the problem described earlier.

- We will study each of the elements used in the *C* code throughout the term.

- By the end of the term you will be able to write programs such as the one used in this example.

Sample C Program (part 2)

```
#include <stdio.h>
#include <stdlib.h>

static int  compute_length (int, int);

int main (void)
{
    typedef struct {
        int         left;
        int         right;
        int         length;
    } seg_t;

    seg_t *seg1, *seg2;
```

library function definitions

main routine, procedure #1

data structure definition

declaration of references to data structure instances

Sample *C* Program (part 3)

```
seg1 = (seg_t *) malloc (sizeof (seg_t));
seg2 = (seg_t *) malloc (sizeof (seg_t));

printf ("Enter left edge of segment 1: ");
scanf ("%d", &(seg1->left));
printf ("Enter right edge of segment 1:");
scanf ("%d", &(seg1->right));
printf ("Enter left edge of segment 2: ");
scanf ("%d", &(seg2->left));
printf ("Enter right edge of segment 2:");
scanf ("%d", &(seg2->right));

seg1->length = computelength (seg1->left,
     seg1->right);
seg2->length = computelength (seg2->left,
     seg2->right);
```

create instances of data structure, and associate with references

input / output, store result in data structure

call a subroutine, store result in data structure

Sample *C* Program (part 4)

```
 if (seg1->length == seg2->length)
  printf("Segment lengths are equal\n");
 else
  printf("Segment lengths are NOT equal\n");

  return 0;
}
```

subroutine, procedure #2

```
int compute_length (int left, int right)
{
  return (right-left);
}
```