

Computer Systems Principles

More on C Programming



Learning Objectives

- **Basic Tools**
 - Learn about Make
 - Learn about Valgrind (pronunciation: val-grinned)
 - Learn about GDB
- **Basic Programs**
 - To learn and apply basic C programs
- **Arrays**
 - Understand the significance of the '\0' character
 - Learn how to define arrays and initializers
- **Strings**
 - Understand that C strings are arrays
- **Functions**
 - Understand function definitions
 - Learn about function declarations and prototypes

Programming Tools

- **Programming Languages**
 - Are tools for extending the functionality of the machine the program executes on.
- **Languages Need Tools**
 - To automate the compilation process (build-time)
 - To detect errors in your programs (run-time)
 - To debug your programs (run-time)

Make

- **A Build Automation Tool**
 - Automatically (re-)compiles your source code into an executable object file.
- **Properties**
 - Compile source code if it hasn't been compiled
 - Re-compile source code if it has been changed
 - Easily extend to include additional source code
 - Clean up build when we are done

Make Basics

- **Make Syntax**

target: dependencies

[**tab**] *system command*

Example:

simple2: simple2.c

gcc simple2.c -o simple2

Running Make

- **What file is used?**
 - Makefile (is the default)
 - Or use `-f` flag to the 'make' command
- **What thing is built?**

The target given on the command line:

```
$ make simple2
```

This will match the rule whose target is “simple2”
and *invoke* its system commands

Examples

- **Makefile**
- **Makefile-01**
- **Makefile-02**
- **Makefile-03**

Valgrind

- **We can detect some errors at compile-time**
 - Use the `-Wall` flag (“Warnings all”)
- **Some errors need to be detected at run time**
 - Uninitialized variables/memory
 - Invalid access to memory
 - Memory leaks
- **Valgrind to the rescue!**
 - Valgrind can give you information about errors in your program similar to how Java provides a “stack trace” when an exception occurs!

Running Valgrind

- **First...**
 - Compile your program using make
- **Next...**
 - Run valgrind on your executable:

```
$ valgrind ./simple2
```

GNU Debugger (GDB)

- **Compile-time**
 - gcc with `-Wall` is good!
- **Run-time**
 - valgrind is also good!
- **Interactive Run-time**
 - GNU Debugger (gdb)
 - Lets you explore the program as it executes
 - Helps you detect difficult to find bugs

Running GDB

- **It is as simple as this:**

```
$ gdb ./simple2
```

Let us look at some C programs!

- **simple.c** – we saw this last time
- **simple2.c**
- **simple3.c**
- **echo.c**
- **lines.c**
- **wc.c**

Arrays in C

- **Declaration in C:**

type id[size];

- **Example:**

```
int days[7];
```

```
days[0] = 1; days[1] = 0; days[2] = 0; ...
```

Array Initializer

- **Declaration in C:**

*type id[size?] = { v0, v1, v2, ..., vk };
(where $k \leq \text{size}-1$)*

- **Example:**

```
int days[] = { 1, 0, 1, 1 };
```

Array Example

- **arrays.c**

Array Bounds

- **What does Java do?**
 - It has array bound checking; i.e., it will throw an exception if you run off the end of an array.
 - This is good.
- **What does C do?**
 - Nothing!
 - If you run off the end of an array - **you are on your own!**
 - One of *the largest* sources of bugs and security holes in C programs!

Character Arrays

- **Same as other arrays...**

*type id[size?] = { v0, v1, v2, ..., vk };
(where $k \leq \text{size}-1$)*

- **Example:**

char name[3]; name[0] = 'T'; name[1]='i'; ...

OR

char name[] = { 'T', 'i', 'm' };

Character Arrays as Strings

- **What about strings in C?**
 - Turns out that C does not have a type for strings!
 - Instead, we use an array of characters...but
 - We need to use `'\0'` (null) character to terminate!
- **Example:**

```
char name[] = {'T', 'i', 'm', '\0'};
```

OR

```
char name[] = "Tim"; // compiler supplies \0!
```

String Example

- **strings.c**

Multi-dimensional Arrays?

- **Yes! And it is easy:**

```
int matrix[10][10];
```

- **Assignment is easy:**

```
matrix[0][1] = 55;
```

How about arrays of strings?

- **We have already seen this:**

```
int main(int argc, char *argv[]) { ... }
```

- **What is the ‘*’ business?**

This turns out to be a “pointer”. And to make it more confusing arrays and pointers are very much related...

More on this next time!

Initializing an Array of Strings

// Initialization:

```
char *names[10] = { "Tim", "Caleb", "Hazel", ... };
```

// Printing one string using printf:

```
printf("%s\n", names[0]);
```

Array of String Example

- **string-array.c**

C Functions

- **Basic Structuring Mechanism**
 - In Java we have classes...
 - In C we have functions
- **C functions consist of:**
 - function declarations (**the interface**)
 - function definitions (**the implementation**)

Function Example

- **functions.c**