ECE264: Advanced C Programming

Summer 2019

Week 1: Tools, Program Layout, Data Types and Structs

Git

- Version Control System
  - Manage versions of your code – access to different versions when needed
  - Lets you collaborate
- 'Repository' – virtual storage
  - Local and Remote Repository
    - Local is working copy

Git – Initializing Repositories

- Getting started with local working copies:
  - git init

```
Terminal
[ecegrid-thin4:~/ECE264/dem0] hegden$ls -a
.  ..
[ecegrid-thin4:~/ECE264/dem0] hegden$git init
Initialized empty Git repository in /home/min/a/hegden/ECE264/dem0/.git/
[ecegrid-thin4:~/ECE264/dem0] hegden$ls -a
.  ..  .git
[ecegrid-thin4:~/ECE264/dem0] hegden$
```

  - git clone (when a remote repository on github.com exists)

```
Terminal
[ecegrid-thin4:~/ECE264] hegden$ls -a
.  ..
[ecegrid-thin4:~/ECE264] hegden$git clone git@github.com:ece264summer2019/dem0.git
Cloning into 'dem0'...
warning: You appear to have cloned an empty repository.
[ecegrid-thin4:~/ECE264] hegden$ls -a
.  ..  dem0
[ecegrid-thin4:~/ECE264] hegden$cd dem0/
[ecegrid-thin4:~/ECE264/dem0] hegden$ls -a
.  ..  .git
[ecegrid-thin4:~/ECE264/dem0] hegden$
```

Git – Adding Content

- Staging

```
Terminal
[ecegrid-thin4:~/ECE264/dem0] hegden$echo "This repository is created for demo purposes." > README.txt
[ecegrid-thin4:~/ECE264/dem0] hegden$git add README.txt
```

- Commit (save changes in local repository)

```
[ecegrid-thin4:~/ECE264/dem0] hegden$git commit -m "My first commit"
[master ab680c6] My first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.txt
```

- Save changes in remote repository (guard against accidental deletes)

```
[ecegrid-thin4:~/ECE264/dem0] hegden$git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Writing objects: 100% (3/3), 290 bytes | 145.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:ece264summer2019/dem0.git
   3dccc4f..ab680c6  master -> master
```

Git – Releasing Code

- Tagging
  - make sure there are no unsaved changes in local repository

```
[ecegrid-thin4:~/ECE264/dem0] hegden$git status .
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

```
[ecegrid-thin4:~/ECE264/dem0] hegden$git tag -a RELEASE_V0.1 -m "First release"
```

- Save tags in remote repository

```
[ecegrid-thin4:~/ECE264/dem0] hegden$git push --tags
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 176 bytes | 176.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To github.com:ece264summer2019/dem0.git
 * [new tag]         RELEASE_V0.1 -> RELEASE_V0.1
```

Git – Recap..

- Please read https://git-scm.com/book/en/v2 for details

```
1.  git clone (creating a local working copy)
2.  git add (staging the modified local copy)
3.  git commit (saving local working copy)
4.  git push (saving to remote repository)
5.  git tag (Naming the release with a label)
6.  git push --tags (saving the label to remote)
```

Makefile

- Is a file, contains instructions for the ‘make’ program to generate a target (executable).
- Generating a target involves:
  1. Preprocessing (e.g. strips comments, conditional compilation etc.)
  2. Compiling (.c -> .s files, .s -> .o files)
  3. Linking (e.g. making printf available)
- A Makefile typically contains directives on how to do steps 1, 2, and 3.

Makefile - Format

- Contains series of ‘rules’-

```
target: dependencies
[TAB]  system command(s)
```

Note that it is important that there be a TAB character before the command (not spaces).

Example,

```
testgen: testgen.c
	gcc testgen.c –o testgen
```

- And Macro/Variable definitions -

```
CFLAGS = -std=c99 -g -Wall -Wshadow --pedantic -Wvla –Werror

GCC = gcc
```

Makefile - Usage

- The 'make' command (Assumes that a file by name 'makefile' or 'Makefile'. exists)

```
[ecegrid-thin4:~/ECE264/dem0] hegden$cat makefile
GCC=gcc
CFLAGS=-std=c99 -g -Wall -Wshadow --pedantic -Wvla -Werror
testgen: testgen.c
	$(GCC) $(CFLAGS) testgen.c -o testgen
clean:
	rm testgen
[ecegrid-thin4:~/ECE264/dem0] hegden$make
gcc -std=c99 -g -Wall -Wshadow --pedantic -Wvla -Werror testgen.c -o testgen
[ecegrid-thin4:~/ECE264/dem0] hegden$
```

- To know more, please read: https://www.gnu.org/software/make/manual/html_node/index.html#Top

Sorting

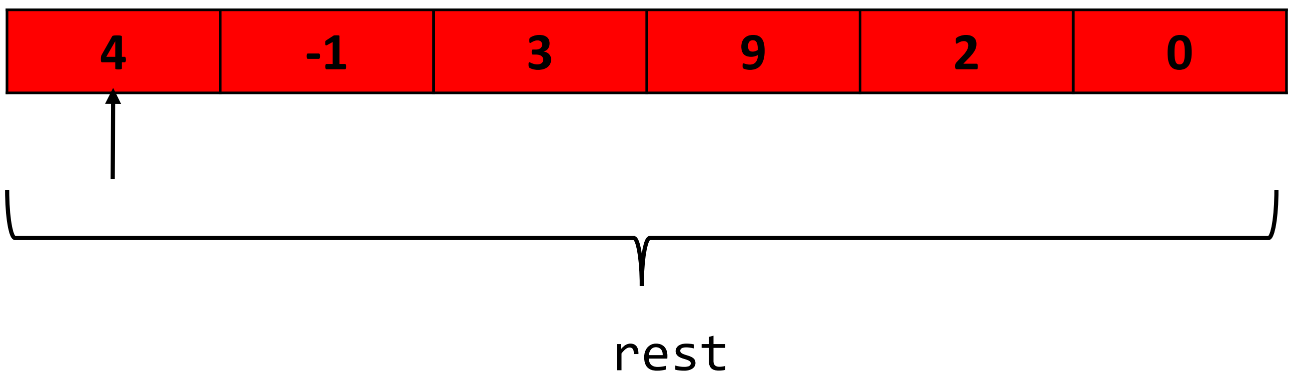
- Arranging the elements of a list in a particular order.
- E.g. sorting list of names in lexicographical order, sorting numerical input in ascending order, etc.
- Used often as a pre-processing step in optimizing computation.
  - Easier and faster to locate items

Sorting - Selection sort

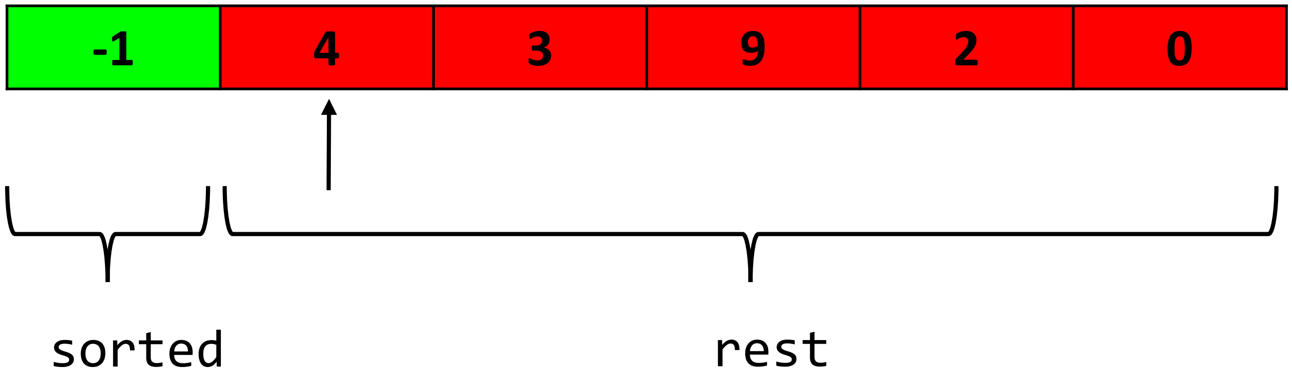
- Repeatedly find the minimum element in the unsorted array and put it at the beginning.
  - Divides the input array into 2 pieces - `sorted` and `rest`.
  - *All elements* in `sorted` are smaller than *any element* in the `rest` – *invariant*
  - Works by growing `sorted` and shrinking `rest`

Selection sort - example

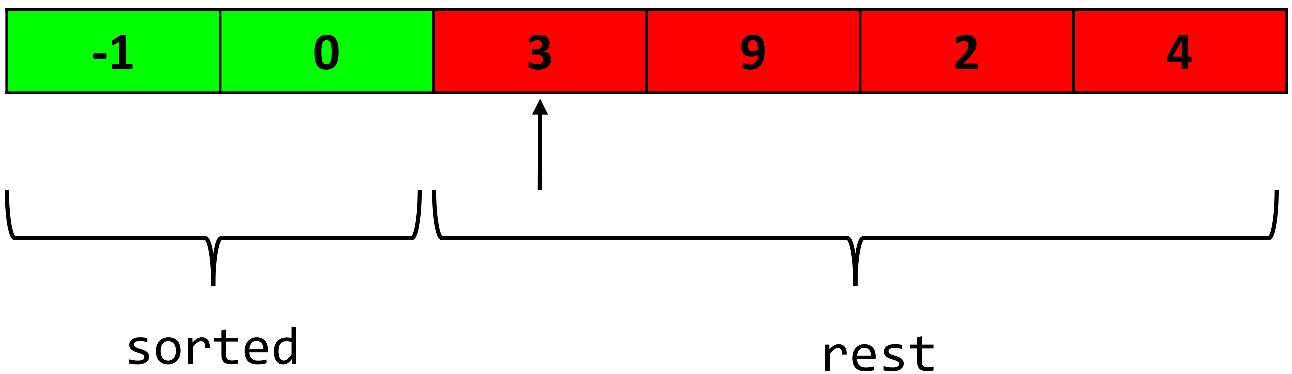
- A cursor dividing `sorted` and `rest`



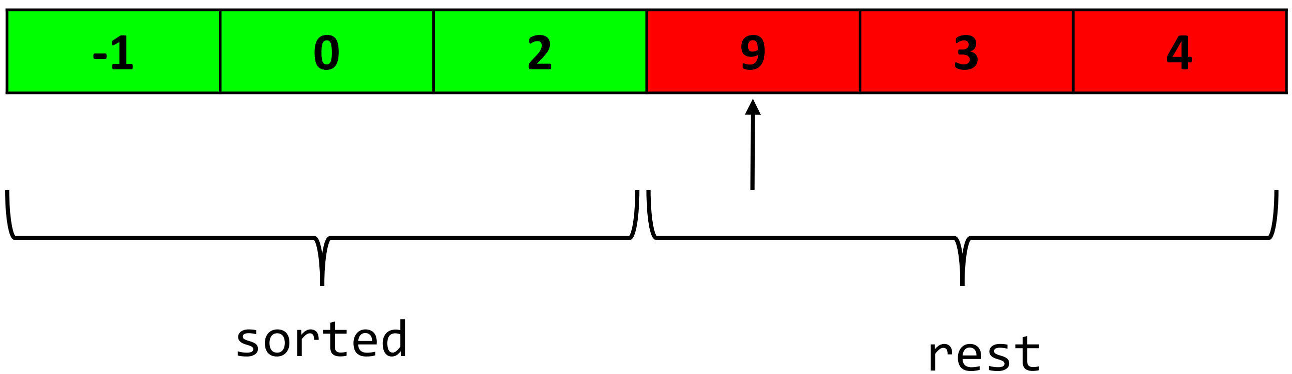
Selection sort - example



Selection sort - example



Selection sort - example



Selection sort - example

-1	**0**	**2**	**3**	**4**	**9**

sorted

Sorting algorithms - Evaluation

- Many metrics used for evaluating sorting algorithms.
- Two most common metrics are:
  - How many comparisons are involved?
  - How much data movement is involved?

Selection sort - pseudocode

```
int input[N] = //input
int cursor = 0 //initial position of the cursor
for(cursor = 0; cursor < N; cursor++)
 //sorted list from [0,cursor)
 //rest of the list from [cursor, N)
 for(i = cursor; i < N; i++)
   //search the rest of the list to find the smallest value
 //swap the smallest value with the value at input[cursor]
```

Selection sort - Analysis

```
1 int input[N] = //input
2 int cursor = 0 //initial position of the cursor
3 for(cursor = 0; cursor < N; cursor++)
4  //sorted list from [0,cursor)
5  //rest of the list from [cursor, N)
6  for(i = cursor; i < N; i++)
7    //search the rest of the list to find the smallest value
8  //swap the smallest value with the value at input[cursor]
```

- Outer loop (line 3) is moving the cursor, inner loop (line 6) is finding minimum.

How many times does inner loop execute?

Selection sort - Analysis

```
int input[N] = //input
int cursor = 0 //initial position of the cursor
for(cursor = 0; cursor < N; cursor++)
 //sorted list from [0,cursor)
 //rest of the list from [cursor, N)
 for(i = cursor; i < N; i++)
   //search the rest of the list to find the smallest value
 //swap the smallest value with the value at input[cursor]
```

- inner loop runs N times, (N - cursor) iterations every time.

$$= \sum_{i=0}^{N-1} N - i$$

$$= \sum_{i=1}^{N} i \qquad = \frac{N\,(N+1)}{2}$$

Selection sort - Analysis

- outer loop runs for N iterations
- inner loop runs for ~ N(N+1)/2 iterations
  - inner loop dominates

1. Approximately how many array write operations occur?

2. Double the input, how long does Selection sort take?

Number Bases

- We use decimal (base-10), Computers use binary (base-2).
- Binary is difficult to read. So, we use Hexadecimal (base-16).
- Octal (base-8) is the other popular number format.

Number Bases - Hexadecimal

- Hexadecimal uses 16 digits: 0 to 9 and A to F. A to F represent decimal numbers 10 to 15.
- A digit in hexadecimal needs 4 bits. Therefore, a byte of information (8 bits) represents two digits.
- Example:

Decimal	Binary	Hexadecimal
10	1010	0xA
16	1 0000	0x10
43981	1010 1011 1100 1101	0xABCD

How are Numbers Stored in Memory? - Endianness

- Assume an integer needs 4 bytes of storage
  - E.g. 1193 in Hexadecimal = 0x4A9 = 0x 00 00 04 A9 when stored in 4 bytes of memory.
  - How are those 4 bytes ordered in memory? – Endianness
- Two popular formats: Big-Endian and Little-Endian

Big-Endian

- Most-significant-byte (MSB) at low-address and least-significant-byte (LSB) at high-address
  - E.g. 1193 = **0x00 00 04 A9** (= 4 * 16^2 + A * 16 + 9)
  - MSB (0x00) is written at lower address, LSB (0xA9) is written at higher address.

	0000 0000 **(00)**	0000 0000 **(00)**	0000 0100 **(04)**	1010 1001 **(A9)**
Address:	0x00000001	0x00000002	0x00000003	0x00000004

  - Motorola 68000 Series, IBM-Z Mainframes.

Little-Endian

- Most-significant-byte (MSB) at high-address and least-significant-byte (LSB) at low-address
  - E.g. 1193 = **0x00 00 04 A9** (= 4 * 16^2 + A * 16 + 9)
  - MSB (0x00) is written at higher address, LSB (0xA9) is written at lower address.

	1010 1001 **(A9)**	0000 0100 **(04)**	0000 0000 **(00)**	0000 0000 **(00)**
Address:	0x00000001	0x00000002	0x00000003	0x00000004

  - Intel x86 Architecture

Little-Endian

- What gets flipped in Little-endian?

Flipped	**Not-Flipped**
• Bytes • Multi-byte numbers (e.g. int, long, float) and addresses	• Bits within a byte, Hex-digits within a byte • Array elements and Struct fields

Endianness

- Fortunately, we don’t have to worry about endianness.
  - You don’t have to reverse bytes when you read an integer.
  - Compiler and the processor do the job for you.
- However, you need to be aware of endianness when inspecting memory contents.
  - E.g. when using GDB while debugging.

Program Layout in Memory

- Why know it?
  - Debug programs
  - Design software for constrained devices (e.g. embedded systems)
  - Design robust (secure) software

Program Layout in Memory

- A program's memory space is divided into four segments:

  1. Text
     - source code of the program
  2. Data
     - Broken into uninitialized and initialized segments; contains space for global and static variables. E.g. `int x = 7; int y;`
  3. Heap
     - Memory allocated using malloc/calloc/realloc
  4. Stack
     - Function arguments, return values, local variables, special registers.

Detour - Stacks

Real Stack

Hardware Stack

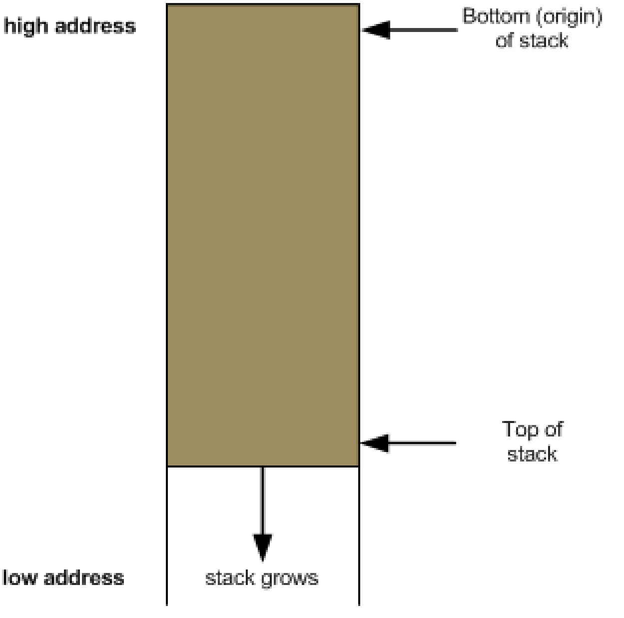
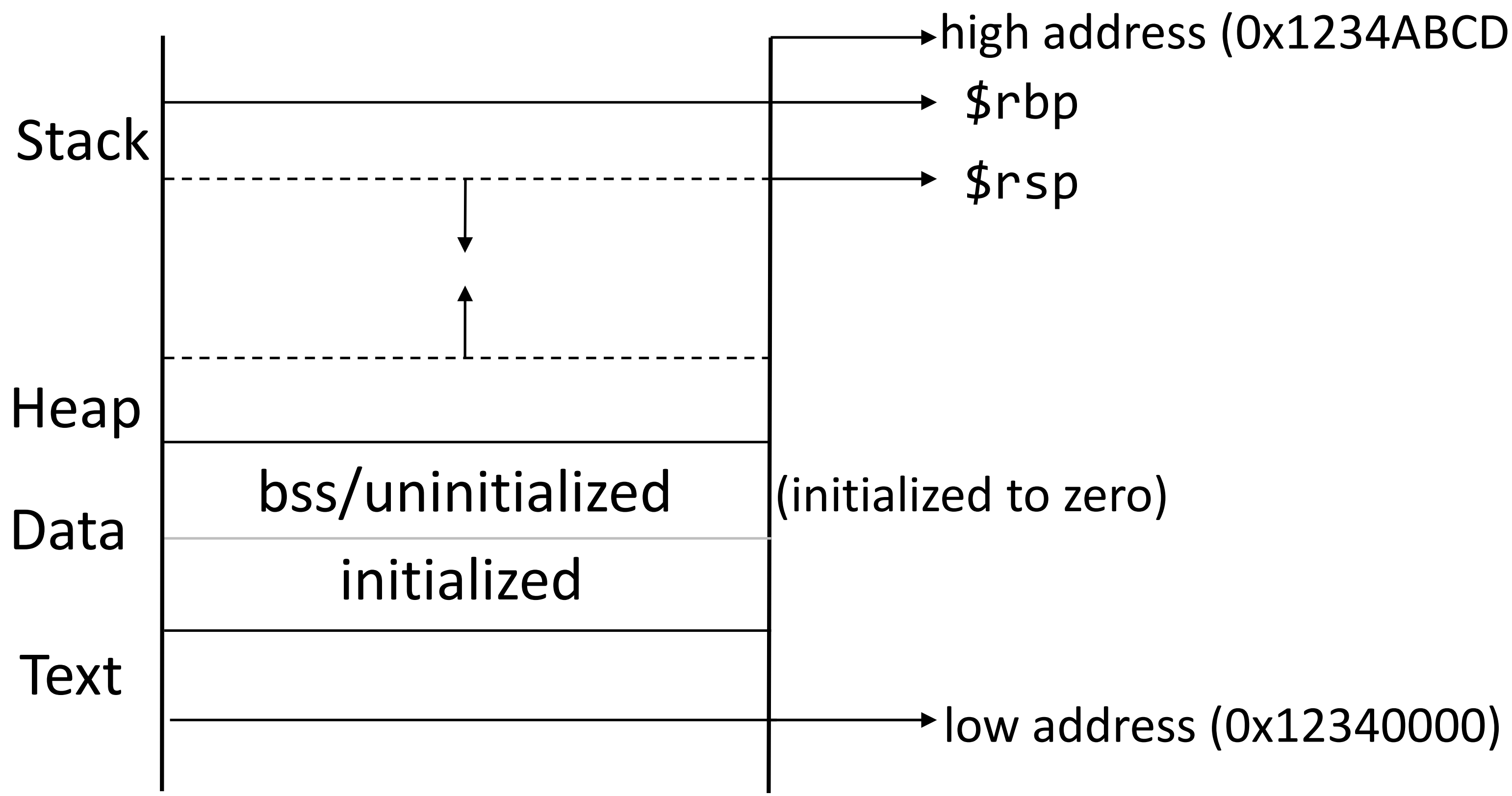


Image source: https://eli.thegreenplace.net/2011/02/04/where-the-top-of-the-stack-is-on-x86/

Stack Frame

- A sub-segment of memory on the stack space
  - Special registers `$rbp` and `$rsp` track the bottom and top of the stack frame.
  - Example: when `main` calls function `foo`
    1. The following are pushed on to stack:
       - `foo`'s arguments
       - Space for `foo`'s return value
       - Address of the next instruction executed (in `main`) when `foo` returns
       - Current value of `$rbp`
    2. `$rsp` is automatically updated (decremented) to point to current top of the stack.
    3. `$rbp` is assigned the value of `$rsp`

Program Layout in Memory



Question ?

Where are the command-line arguments stored?

GDB

- GNU Debugger – A tool for inspecting your C programs
  - How to begin inspecting a program using gdb?
  - How to control the execution?
  - Misc – displaying stack frames, visualizing assembler code.
  - How to display, interpret, and alter memory contents of a program using gdb?