Computer Architecture
CS-211

Spring 2017

Recitation #2

Agenda

- Assignment 1 Setup
  - Autograder
  - Pass argument to C program
  - malloc a 2D array
  - BST deletion
- Makefile
- How to debug with gdb

How to Tar/Untar

- How to tar:

Go to the parent directory of pa1

  $ **tar cvf pa1.tar pa1**

Another example

  $ **tar cvf my new.tar dir1 dir2 le1 le2**

- How to untar:

Go to the directory where you have .tar le

  $ **tar xvf pa1.tar**

Makefile

- Makefiles are a simple way to organize code compilation.
- Why we need makefile
  - Hard to enter long command to compile each time
  - Only compile the code that has been changed! (Take less time)
- A complete reference for writing ‘makefile’ from basic one to very advanced one here: http://www.gnu.org/software/make/manual/make.html
- It consists of set of rules

variable definitions e.x.

Target: dependency1 dependency2 …

<tab> action

Help on Makefile

Youtube : https://www.youtube.com/watch?v=kGGE8mtrbrM Blog :http://mrbook.org/blog/tutorials/make/

- Makefile doesn't have any file extension.
- Default name of the Makefile for any project is Makefile (otherwise you need to specify the name like: $make -f mymakefile)
- TAB must be placed before each command written in Makefile
- If you do not place TAB correctly in your Makefile, you may encounter errors.
- You can check the TAB key in any editor.
- In Linux/Mac, open your Makefile in vim editor then press ESC then type "set list" and hit ENTER. It should show the TAB as "^I".
- In Windows, open your file using Notepad++ editor. Then go to View --> Show Symbol --> Show All Characters. In place of TAB, you should see "-------->". If you don't see this characters in place of TAB, then you should correct your makefile.

Run autograder

```
autograder
  pa1
    |-  first
      |--  first.c
      |--  first.h (if used)
      |-- Makefile
    |- second
      |-- second.c
      |-- second.h (if used) .
      |-- Makefile
```

Run autograder2

Untar autograder.tar

$ tar xvf autograder.tar

Copy pa1 into autograder directory

$ cp -r pa1 autograder

Go to autograder directory

$ cd autograder

Run autograder

$ python auto_grader.py

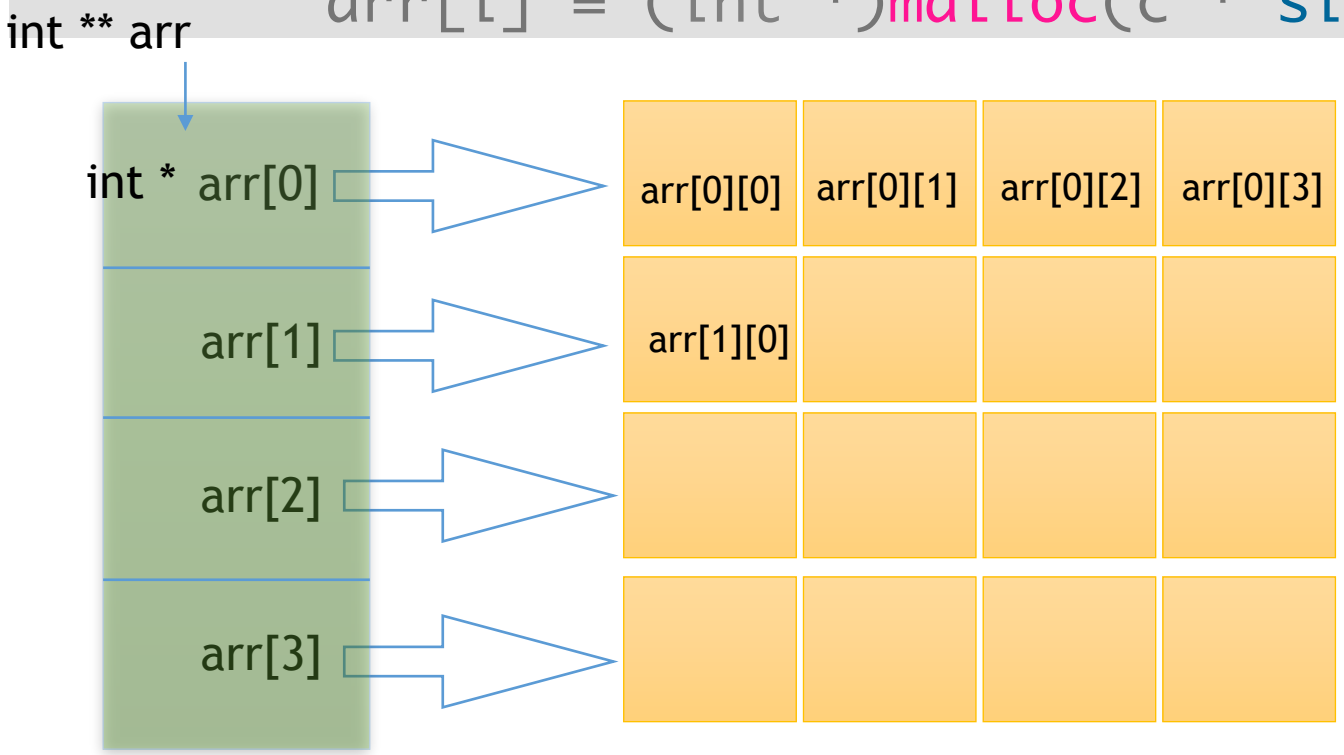
Passing argument to C program

```
int main(int argc, char *argv[]) {
    if(argc <= 1) {
        printf("You did not feed me arguments");
        return 1 ;
    }  //otherwise continue on our merry way....
    int arg1 = atoi(argv[1]);  //argv[0] is the program name
                               //atoi = ascii to int

}
```

allocating a 2D array

```
int **arr = (int **)malloc(r * sizeof(int *));
    for (i=0; i<r; i++)
        arr[i] = (int *)malloc(c * sizeof(int));
```



gdb

Provides extensive facilities for tracing / altering program execution

- Step through program line at a time
- Monitor / modify internal variables
- Call functions independently of normal behavior
- Reversible debugging – step program backward

You need to compile your code with -g :

```
gcc -g ave.c
```

http://www.yolinux.com/TUTORIALS/GDB-Commands.html#GDB_COMMAND_LINE_ARGS

- run <command line argument>: Start program execution from the beginning of the program
- break <line number>
- next (will not enter function)
- step (step inside the function)
- continue
- list <line number or function name>
- print <variable name > or print <variable> = new_value
- info local
- delete (delete all the break points)
- clear <line number>
- reset
- quit

Next time

- pointer example
- link list