CSCE 313
LAB session

Lab policies

- You can choose any of the following for code compiling
  - Ubuntu (preferably, some lightweight versions of it, e.g., Xubuntu) VM
  - Ubuntu Dual boot
  - Amazon Web Services
  - ~~Department servers (linux, compute, build…)~~
- Online Repository
  - We highly recommend using github because of so many past instances of students loosing their code
    - We will not allow you more time for loosing code
  - Make sure to mark your code repo for this course as **"Private"** at all times (note that the default option is public)

Grading policies

- Programming Assignments (PAs)
  - Individual work
- Grade Allocation: 50%
- Late Submission Penalty
  - 12% after a day (0.5% per hour)

Programming Assignments (PAs)

- Total 6 (+/- 1) PAs of varying complexity
- Some PAs may also allocate opportunities for bonus points
- PAs are designed to stay in sync with subject matter covered in class
- PA0 is getting the dev env ready
- PA1 is designed to get C/C++ memory allocation, pointer arithmetic, etc. brushed up
- Rest of the PAs leverage class topics

PA Code Submission

- Students must strictly follow the instructions provided in PA handouts
- All submission are in **Ecampus**
  - We have not adopted Canvas
- Must make a youtube video demoing your work and submit the link along with your work
  - Do not upload the video to ecampus, that makes the submission too big
  - Without the video, you loose 20% points by default, unless otherwise specified (e.g., PA0 does not a video)
  - If you finish before deadline and can demo to your TA during office hours or lab meetings, you may be waived the video submission requirement
- **Plagiarism**
  - **We will run plagiarism checks through MOSS**
    - **We will use all previous PA submissions to this course**
  - **All plagiarism cases (even suspected ones) will be directly reported to the honor office**

PA Help Resources

1. Valgrind and gdb debugging tools
   - https://valgrind.org/docs/manual/quick-start.html
   - https://www.cs.cmu.edu/~gilpin/tutorial/
2. Lab meetings
   - Most effective (face-to-face via Zoom)
3. Office hours of TAs
   - All over the week
   - You can contact TAs from other sections, look them up in the class website
4. Piazza Discussion
   - Frequently Asked Questions (1)
   - New Questions/Discussion (2)

~~5. Email~~

- ~~Hard for the teaching staff, when helping with code~~
- ~~Use it as the last resort~~

Questions?

Required Background

- C/C++ language
  - Basic variables: **char** (8 bits), **short** (16), **int** (32), **long** (64), (**unsigned** keyword)
    - sizeof(int)
    - int a; long b; unsigned char c;
  - Control: **if** … **else** (**if**)…, **switch** … **case** … **default**, **for** and **while** loop
    - dead loop: while(1) or for(;;);
  - Array[]
    - char a[10], where a is the address of the array.

Background

- C language
  - Pointer* (store the address)
    - int* p
    - & to get the address, e.g. int a=1; int* p = &a
    - * to access the data in that address, e.g. *p = 2; so now a=2.
    - Pointer can point to a function, e.g. void (*p)(int, char*) defines a pointer to a function like void func(int a, char* b); //useful in Linux kernel development
    - int (*p)[4] vs int *p[4]?
    - Now we have an array like “int a[6]”, how to define a pointer to it, such than we can use the pointer to read/write it?
    - Now we have an array like “int b[6][4]”, how to define a pointer to it, such than we can use the pointer to read/write it?
    - int *p = a;
    - int *p = b, or int (*p)[4] = b; (*(p+1))[2] = 3 or p[1][2] = 3;
      // p+1 means that the address + 4*sizeof(int)
    - Double pointer int **p, a pointer to a pointer. When to use?

Double-Pointer Example

- Array of strings can be implemented
  - e.g int main(int argc, char** argv) { return 0; } ./test a bb ccc → argv[0]="a", argv[1]="bb", argv[2] = "ccc"
- Double-pointer is useful when the pointed variable needs to be changed in other places (e.g. function)
  - Node Insertion in Tree-like data structure, and memory deallocation …

Double pointer approach

```
void insert(BinaryTreeNode **node, int data) {
      if (*node == NULL) {
            *node = getNewNode(data);
      }

      if (data == (*node)->data) {
            //do  nothing
      } else if (data < (*node)->data) {
            insert(&(*node)->left, data);
      } else {
            insert(&(*node)->right, data);
      }
}
```

Single pointer approach

```
BinaryTreeNode* insert(BinaryTreeNode *node, int data) {
      if (node == NULL) {
            node = getNewNode(data);
            return node;
      }

      if (data == node->data) {
            //do  nothing
      } else if (data < node->data) {
            node->left = insert(node->left, data);
      } else {
            node->right = insert(node->right, data);
      }

      return node;
}
```

```
void safeFree(void** memory) {
      if (*memory) {
            free(*memory);
            *memory = NULL;
      }
}
```

```
void* myMemory = malloc(…);

Computation using myMemory

safeFree(&myMemory);
```

Background

- C language
  - –Struct
    - struct student{ int netid; char name[64] ; …};
  - –Function()
    - int func(int a, char* b) { return 0; } //note, function name is also the address of the function, so we can let p = func;
    - main(); //entry point of the program
    - printf("some string %d\n", aaa); //aaa is an integer variable
  - –Definition
    - #define MAX 100 //note, there is no semicolon here
  - –Typedef
    - typedef unsigned char byte;
    - typedef struct student student_t;
    - typedef void (*FUNC)(int, char*); //so FUNC f defines a function pointer.
  - –Comment
    - /* */ cannot be nested
    - //
    - #if 0 … #endif

Background

- C++ language background
  - Class
    - Constructor/Destructor
    - Public/Private/Protected Scope
    - Inheritance of c++
    - Static/Const functions/variables
  - Dynamic Memory Allocation
    - new / delete operators
      - e.g. int *foo = new int [1024];
      - e.g. delete foo; or delete[] foo ??
  - Function Overriding
    - virtual function, pure virtual function?
    - virtual destructor?
  - Others
    - Compile Time Polymorphism
      - e.g. function/operator overloading
    - exception handling, STL libraries
    - template

Background

- Compiler (compile and link)
  - gcc/g++: C compiler/C++ compiler
  - gcc/g++ -o exename file.c , only one source file
  - more than one source files
    - gcc/g++ -c -g file1.c (compile) //-g is adding debug information
    - gcc/g++ -c -g file2.c (compile)
    - gcc/g++ -o exename file1.o file2.o (link)
- Makefile
  - Automatically call the instructions above in a smarter manner (incremental compile).
    - make -f makefile_name
  - automake, cmake, scons etc are tools for auto-generating makefile.
- Debugger
  - Gdb, valgrind, Clion/(or some other) IDE debugger interface

Background

- Bash (Command line environment on Unix/Linux)
  - Use “command --help” to get the command options
  - ls (-l) (list)
  - cd (change directory)
  - pwd (print working directory)
  - ps -ef (to get the #pid of a process, i.e. a running program)
  - kill #pid
  - vi, emacs (terminal based editor)
  - make
- IDE
  - Clion
  - Visual Studio Code (not Visual Studio, both in Win and Linux)

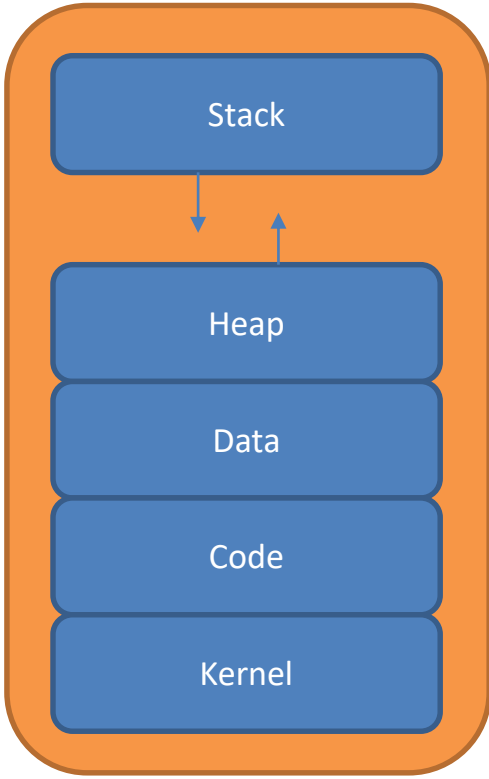
Some Pointer Syntax in C/C++

Let us take a closer look at the following program:

```
datatype x;  // declaring variable x of type datatype in the stack
int x, char x, double x, myowntype x;

datatype* y = &x;  // making a pointer to the variable x

datatype* z = new datatype; // making a datatype instance in heap and
keeping the pointer in z

datatype* w = new datatype [10]; // making an array of 10 datatype
instances in heap and keeping the pointer in w
```

Memory Layout of a Program

- This is the general layout
  - Varies across systems (e.g., kernel is at top in some cases)
  - Another variation comes from "Address Space Layout Randomization" (ASLR) policy to evade attackers
- Let us now see how variables are placed in the address space
- First, make sure to turn off ASLR by running following command:




```
sudo sysctl kernel.randomize_va_space=0
```

Exploring Placement in Address Space

```
void function1(){
        int x = 5;
        int y = x + 15;
        cout <<"Address of x, y: " << &x << ", " << &y << endl;
}
int main (){
        char a, b;
        //printf ("Address of vars: %p %p\n", &a, &b);
        cout << "Address of a, b: " << (void*) &a << ", " << (void*) &b << endl;
        int var1 = 1, var2 = 2;
        cout << "Address of var1, var2: "<< &var1 << ", " << &var2 << endl;
        function1();
        int * hv = &var1;
        cout << "Addr of hv: " << &hv << ", content of hv: " << hv << endl; // hv itself is still in the stack,
        // allocate a variable in heap and put the addr in ha's content
        hv = new int;
        *hv = 5;
        cout << "Addr of hv: " << &hv << ", content of hv: " << hv << ", derefencing hv: " << *hv << endl;
        // let us allocate another variable in the heap and put the address in hv2
        int* hv2 = new int;
        cout << "Addr of h2: " << &hv2 << ", content of h2: " << hv2 << ", derefencing h2: " << *hv2 << endl;
}
```

```
Address of a, b: 0x7fffffffe0df, 0x7fffffffe0de
Address of var1, var2: 0x7fffffffe0e0, 0x7fffffffe0e4
Address of x, y: 0x7fffffffe0b0, 0x7fffffffe0b4
Addr of hv: 0x7fffffffe0e8, content of hv: 0x7fffffffe0e0
Addr of hv: 0x7fffffffe0e8, content of hv: 0x55555556b2c0, derefencing hv: 5
Addr of h2: 0x7fffffffe0f0, content of h2: 0x55555556b2e0, derefencing h2: 0
```

First address in the stack

Second item is just 1 byte off

Integers are 4 bytes off

Function stack is a whole new block, starting below the main's stack

Heap starts far off (i.e., lower address) from the stack

Exploring Layout

- With the numbers in the previous page, we can now annotate different addresses in the layout picture:

```
Address of a, b: 0x7fffffffe0de, 0x7fffffffe0df
Address of var1, var2: 0x7fffffffe0e0, 0x7fffffffe0e4
Address of x, y: 0x7fffffffe0b0, 0x7fffffffe0b4
Addr of hv: 0x7fffffffe0e8, content of hv: 0x7fffffffe0e0
Addr of hv: 0x7fffffffe0e8, content of hv: 0x55555556b2c0, derefencing hv: 5
Addr of h2: 0x7fffffffe0f0, content of h2: 0x55555556b2e0, derefencing h2: 0
```