CS 261-020
Data Structures

Lecture 2
C Basics
1/6/22, Thursday
Odds and Ends

• Due 1/9 Sunday 11:59pm: Quiz 1
• Assignment 1 is posted
Lecture Topics:

• C Basics
C Basics – printf()

• How to print the content of a variable?
  • Passing a **format string** and accompanying arguments to `printf()`
    • *Format string*: a template for the text to be printed. Contains *format specifiers* into which specific value will later be inserted
    • *Format specifier*: start with a %, followed by a character describing the data

• E.g.:
  ```c
  int x = 8;
  printf("This is the value of x: %d\n", x);
  ```
C Basics – scanf()

• How to accept input from standard input (keyboard)?
  • In C++, we use cin
    • i.e., cin >> var;
  • In C, we use scanf()
    • i.e., scanf("%d", &var);

• To read in more than one value, use multiple format specifiers
  • i.e.,
    printf("Enter two integers: \n");
    scanf("%d %d", &var1, &var2);
C Basics – Functions (cont.)

• Unlike C++, C has no reference types!
• Can only pass by value (or by pointers)

```c
#include <stdio.h>

void foo(int *x) {
    printf("foo was passed this argument: %d\n", *x);
}

int main(int argc, char** argv) {
    int val = 5;
    foo(&val);
}
```

• Demo...
C Basics – Structures

• Unlike C++, C has no classes or class functions!
  • C++ is object oriented
  • C is procedural

• Use struct type to represent structured data in C
  • E.g., in C++, we might do:
    ```
    Student s = new Student ("Harry Potter");
    s.print();
    ```
  • In C, we’d do:
    ```
    struct Student s = {.name = "Harry Potter"};
    print_student (s);
    ```
C Basics – Pointers

• A pointer is a variable whose value is a memory address
• Every pointer points data of a specific data type
  • E.g.,
    ```c
    int var = 20;
    int *var_ptr = &var;
    ```
• Demo…
C Basics – Void Pointers (void*)

• A void pointer is a pointer represented by the type `void*`.
• A void pointer is a generic pointer, it can point to data of any data type.
  • E.g., a void pointer points to an integer
    ```c
    int var = 20;
    void *v_ptr = &var;
    ```
  • Can we use a `float*` instead of `void*`?
    • It gives us a warning...
  • Can use `void*` to point to any other type:
    ```c
    float pi = 3.1415;
    struct Student s = {.name = "Harry Potter"};
    v_ptr = &pi;
    v_ptr = &s;
    ```
C Basics – Void Pointers (void*) (cont.)

• Void pointers **cannot be dereferenced directly** since there is no type information
  - E.g.

  ```c
  struct student s = { .name = "Harry Potter" }
  void* v_ptr = &s;
  printf("%s\n", v_ptr->name); /* Compile-time error: can't dereference void pointer */
  ```

• To dereference it, we need to move it back to a pointer variable of the correct type
  - E.g.

  ```c
  struct student* s_ptr = v_ptr;
  printf("%s\n", s_ptr->name);
  ```

  OR Cast it back

  ```c
  printf("%s\n", ((struct student*)v_ptr)->name);
  ```
• Why `void*`?
  • It allows the data structures to contain data of any type while remaining type agnostic

• Demo...
C Basics – Program Memory (stack vs. heap)

• Stack: a small, limited-size chunk of memory from the larger blob of system memory
  • Stores local variables declared in functions,
  • Allocated at compile time, known as statically allocated memory
  • At most 8kb

• Heap: comprises essentially all the rest of system memory
  • A program must make requests to allocate memory from the heap
  • Allocated at runtime, known as dynamically allocated memory
C Basics – malloc()

• Allocating memory on the heap
  • In C++: use `new` operator
  • In C: use `malloc()` ← requires `#include <stdlib.h>`

• `malloc()`:
  • Allocates a contiguous block of memory
  • Arguments: number of bytes
  • Return: `void*`

```c
void * allocated_memory = malloc(NUMBER_OF_BYTES);
```
How to figure out how many bytes to allocate?
- Use `sizeof()`!
- `sizeof()` – returns the size in bytes of a given variable or data type
- E.g., `sizeof(int)` returns 4

Q: How to allocate an array of 1000 integers on the heap?
- `int* array = malloc(1000 * sizeof(int));`
- `int* array = new int[1000];`
C Basics – malloc() and struct

• Use malloc() with struct:
  • struct Student *s = malloc(sizeof(struct student));

• To access the struct’s fields using the pointer:
  (*s).name = “Harry Potter”;
  (*s).gpa = 4.0;
  OR
  s->name = “Harry Potter”;
  s->gpa = 4.0;

• To allocate an array of structs:
  • struct Student* students = malloc(1000 * sizeof(struct student));
C Basics – Free dynamic memory

• We have to **manually free** memory allocated on the heap
  • otherwise → memory leak!

• How?
  • In C++, we use `delete`
  • In C, we use `free()`
  • E.g.,
    ```
    int* array = malloc(1000 * sizeof(int));
    ...
    free(array);
    array = NULL;
    ```

• Rule of thumb: For every call to `malloc()` you should have a corresponding call to `free()`