CS232L Operating Systems Lab
Lab 05: Introduction to C Programming - Part 2

CS Program
Habib University

Fall 2024

1 Introduction

This document assumes that you are already familiar with C++, and thereby with the basics of C.

In this lab you will learn how to:

1. Split your program in multiple files
2. Do Dynamic memory management
3. Do basic use of makefiles

2 C Programming Workflow

The basic flow of writing and executing a C program is [2]:

3 Resources

The course book comes with a tutorial for lab which introduces the basics of using **gcc** and **makefiles**. The tutorial is available at the link:
http://pages.cs.wisc.edu/ remzi/OSTEP/lab-tutorial.pdf

4 Dynamic memory management

C provides functions to allocate memory from the `heap` dynamically during run time. Some of them are:

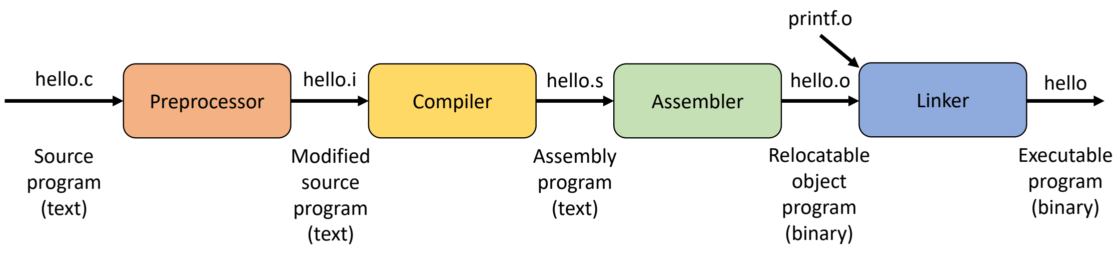

Figure 1: The C compilation process.

```c
# include <stdlib .h>
void * malloc ( size_t size );
void free ( void * ptr );
void * calloc ( size_t nmemb , size_t size );
void * realloc ( void * ptr , size_t size );
void * aligned_alloc ( size_t alignment , size_t size );
```

Listing 1: Heap functions

We will see the usage of `malloc()` and `free()` in this lab.

The caller passes the size of memory they want to allocate (in bytes) to `malloc` and `malloc`, if successful, would return a pointer to the start of memory segment just allocated. If unsuccessful, it returns NULL.

The return type `malloc` is `void*` i.e. a pointer of an unknown type. That is because `malloc` does not know what will this memory be used for. The calling function can assign this memory address to a pointer of desired type and manipulate the memory via this pointer.

This memory will not be automatically freed. The function `free` is used to return memory back to heap. It takes a pointer containing the address of a memory segment previously allocated by a call to `malloc`. It is the job of the programmer to call `free` when they have no more use of the memory allocated by `malloc`. Failure to do so would result in memory leaks.

```c
unsigned int length = get_student_count();
double * gpa_array = (double *) malloc ( length * sizeof(double) );
for ( unsigned int i = 0; i < length ; ++i) {
  largeVec [i] = 0.0;  //initialize
}
...
// here we use the array
...

free (gpa_array); //don't for get this!!
```

Listing 2: malloc and free

The codes in listings 3, 4, 5 show how to use `malloc` and `free` to create a dynamic queue containing integer elements.

```c
#ifndef MY_Q_H
#define MY_Q_H

struct node {
  int val;
  struct node *next;
};


// enqueue'd at tail
void enqueue (struct node ** headaddr, int val );
// dequeue'd from head
int dequeue (struct  node **headaddr);

void print (struct node * head);
#endif
```

Listing 3: my_q.h

```c
#include <stdio.h>
#include <stdlib.h>
#include "my_q.h"

// enqueue'd at tail
void enqueue (struct node ** headaddr, int val ) {
  if (headaddr==NULL){
    fprintf(stderr, "NULL ptr passed\n"); exit(1);
  }


  struct node * n = malloc(sizeof(struct node));
  if (n==NULL){
    fprintf(stderr, "memory allocation failed\n"); exit(1);
  }
  n->val = val;
  n->next = NULL;


  if( *headaddr==NULL){ // empty list
    *headaddr = n;
  }
  else {
    // get to tail
    struct node *tmp = *headaddr;
    while (tmp->next != NULL)
      tmp = tmp->next;

    tmp->next = n;
  }
}


// dequeue'd from head
int dequeue (struct  node **headaddr) {

  if (headaddr==NULL){
    fprintf(stderr, "NULL ptr passed\n"); exit(1);
  }

  if (*headaddr==NULL) { // list is empty
    return -1;
  }
  else {
    struct node *n = *headaddr;
    *headaddr = (*headaddr)->next;
    int val = n->val;
    free(n);
    return val;
  }
}


void print (struct node * head) {
  if (head==NULL)
    fprintf(stdout, "empty queue\n");

  else {
    while (head!=NULL){
      fprintf(stdout, "%d,", head->val);
      head = head->next;
    }
    fprintf(stdout, "\n");
  }
}
```

Listing 4: my_q.c

```
#include <stdio.h>
#include <math.h>
#include "my_q.h"

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


  struct node *head = NULL;
  fprintf(stdout, "queue status: ");
```

```c
  print(head);

  enqueue (&head, 33);
  enqueue (&head, 55);
  enqueue (&head, 6);
  fprintf(stdout, "queue status: ");
  print(head);

  int val1 = dequeue(&head);
  int val2 = dequeue(&head);
  fprintf(stdout, "queue status: ");
  print(head);

  int val3 = dequeue(&head);
  fprintf(stdout, "queue status: ");
  print(head);

  return 0;
}
```

Listing 5: main.c

4.1 Detecting memory leaks

`valgrind` is a useful tool which can be installed on most unix-like systems. It runs your program and profiles it for memory leaks, giving you a report about potential memory leaks at the end.

You run it by typing:
`valgrind your-exe-name list-of-your-exe-arguments`

5 Makefiles

The **make** utility has been used historically to maintain build systems on unix-like systems.

Please refer to the section **F.6** lab tutorial from OSTEP site for a discussion of the makefiles.

6 Debugging

Debugging is another important skill to master if you intend to make a career in programming. Different techniques and softwares can be used for this purpose.

For now, you can start with one of the most primitive forms, i.e., using print statements. The humble `printf` can take you a long way.

More sophisticated techniques exist of course but we won't be getting into them right now. `gdb` is one the most powerful debuggers out there and comes installed with the `gcc` suite. For the more adventurous among you, the section **F.7** of the lab tutorial on OSTEP site gives an introduction.

7 Exercises

1. Modify the code in my_q.c to make a my_stack.c with `push` and `pop` functions.
2. Follow the code in my_q.c to create a dynamic linked list where you can insert and remove elements even from the middle of it by providing the index (0-based).

3. Write makefiles for above programs with rules for `build`, `rebuild`, `clean`, `run`.

References

[1] Raymond Eric S. *"Basics of Unix Philosophy" The Art of Unix Programming.* Addison-Wesley, Professional.

[2] BTYANT, O'HALLARON. *Computer Systems, A Programmer's Perspective.* Pearson.