Trabajo Practico 0: Infraestructura Basica

Augusto Arturi (#97498)
turitoh@gmail.com

Matias Rozanec (#97404)
rozanecm@gmail.com

Agustin Miguel Payaslian (#96885)
payas17@hotmail.com

Grupo Nro. # - 2do. Cuatrimestre de 2017

66.20 Organización de Computadoras
Facultad de Ingeniería, Universidad de Buenos Aires

7/09/2017

Resumen

El trabajo consiste en programar la Criba de Eratostenes,con el objetivo de familiarizarse con las herramientas que utilizaremos a lo largo del curso.

1. Introducción

Aquí se comenta en forma escueta como está constituido el presente informe, donde básicamente se encuentran dos secciones principales: Desarrollo y Conclusiones.
En Desarrollo se encuentran breves comentarios sobre la implementacion del algoritmo como tambien, las corridas de prueba del programa. En la seccion conclusiones se discuten los resultados obtenidos.

2. Desarrollo

2.1. Implementacion

Pasaje de argumentos: para facilitar el uso de argumentos en el programa, se utilizo la funcion getopt_long() la cual permite utilizar argumentos de forma sencilla.

Criba de Eratostenes: Para la implementacion del algoritmo de la Criba de Eratostenes se siguieron los pasos indicados en [3].

2.2. Corridas de prueba

Compilacion del programa

```
root@:~# gcc -Wall -std=c99 -lm main.c -o erat
```

Mensaje de ayuda y version

```
root@:~# ./erat -h
Usage:
        erat-h
        erat -V
        erat [options] N
Options:
        -h,--help       Print usage information.
        -V, --version   Prints version information.
        -o, --output    Path to output file.
Examples:
        erat -o - 10
root@:~# ./erat -v
Version 1.0
```

Mensaje de ayuda y version

```
root@:~# ./erat -h
Usage:
```

```
        erat -h
        erat -V
        erat [options] N
Options:
        -h,--help       Print usage information.
        -V, --version   Prints version information.
        -o, --output    Path to output file.
Examples:
        erat -o - 10
root@:~# ./erat -v
Version 1.0
```

Corridas con parametros o argumentos incorrectos

```
root@:~# ./erat -o - -5
Given number (-5) is not a valid number
root@:~# ./erat -o - 1
Given number (1) is not a valid number
root@:~# ./erat -o
erat: option requires an argument -- o
Invalid option, run ./erat -h for help
root@:~# ./erat -o -
Given number (0) is not a valid number
root@:~# ./erat
Invalid option, run ./erat -h for help
```

Corridas exitosas

```
root@:~# ./erat -o - 10
2
3
5
7
root@:~# ./erat -o criba 50
root@:~# ./erat -o eratos 100
```

3. Conclusiones

A. Apendice A:Codigo fuente

Archivo main.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <getopt.h>
#include <math.h>
#define MAX_INPUT 1000000


void print_help_data(){
        printf("Usage:\n \terat -h \n\terat -V \n\terat [options] N \
            \n"
                                        "Options:\n\t-h,--help \tPrint \
                                            usage information.\n\t-V, "
```

```
                                        "--version \tPrints version
                                            information.\n\t-o, --output
                                            \t"
                                        "Path to output file.\nExamples
                                            :\n\terat -o -10\n");
}

void print_version_info(){
        printf("Version 1.0\n");
}

void erat(unsigned int * numeros,unsigned int N){

        int i = 2;
        int j = 2;

        while(pow(i,2)<N){
                while(i*j <= N){
                        numeros[i*j] = 0;
                        j++;
                }
                i++;
                j = 2;
        }

}



void print_sieve_of_Eratosthenes(char* path, unsigned int N){

    unsigned int all_nums [N+1];
        for (int i = 0; i <= N; ++i){
                all_nums[i] = i;
        }
        erat(all_nums, N);

        /* check where to print results */
        if (*path == 45){
                // output to stdout
        for (int i = 2; i <= N; ++i){
                if(all_nums[i]!=0)
                        printf("%d\n", all_nums[i]);
        }
        }else{
                FILE* file = fopen(path, "w");
        for (int i = 2; i <= N; ++i){
                if(all_nums[i]!=0)
                fprintf(file, "%d\n", all_nums[i]);
        }
                fclose(file);
        }

}

void check_input_num_and_print_sieve(char* path, int N){
        if (N > MAX_INPUT){
                fprintf(stderr, "Given number (%i) exceeded max
                    input %d.\n", N, MAX_INPUT);
        }else if (N < 2){
                fprintf(stderr, "Given number (%i) is not a valid
                    number\n", N);
```

```
        }else{
                print_sieve_of_Eratosthenes(path, N);
        }
}

int main(int argc, char* argv[]){
        static struct option long_options[] = {
                {"help",        no_argument,            0, 'h' },
                {"version",     no_argument,            0, 'v' },
                {"output",      required_argument,      0, 'o' },
                {0,                     0,
                                    0,  0  }
        };
        int option_index = 0;
        int c = getopt_long(argc, argv, "hvo:", long_options, &
            option_index);
        switch(c){
                case 'h':
                        print_help_data();
                        break;
                case 'v':
                        print_version_info();
                        break;
                case 'o':
                        check_input_num_and_print_sieve(optarg,
                            atoi(argv[argc-1]));
                        break;
                default:
                        printf("Invalid option, run ./erat -h for
                            help\n");
        }
        return 0;
}
```

B. Apendice B:Enunciado

66:20 Organización de Computadoras
Trabajo práctico 0: Infraestructura básica

1. Objetivos

Familiarizarse con las herramientas de software que usaremos en los siguientes trabajos, implementando un programa (y su correspondiente documentación) que resuelva el problema piloto que presentaremos más abajo.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes.

Además, es necesario que el trabajo práctico incluya (entre otras cosas, ver sección 7), la presentación de los resultados obtenidos, explicando, cuando corresponda, con fundamentos reales, las causas o razones de cada resultado obtenido.

El informe deberá respetar el modelo de referencia que se encuentra en el grupo, y se valorarán aquellos escritos usando la herramienta TEX / LATEX.

4. Recursos

Usaremos el programa GXemul [1] para simular el entorno de desarrollo que utilizaremos en este y otros trabajos prácticos, una máquina MIPS corriendo una versión relativamente reciente del sistema operativo NetBSD [2].

5. Programa

Se trata de una versión en lenguaje C de la Criba de Eratóstenes [3]. El programa recibirá por `stdin` un número natural N, y dará por `stdout` (o

escribirá en un archivo) una lista de todos los números primos menores que N. De haber errores, los mensajes de error deberán salir exclusivamente por `stderr`.

5.1. Comportamiento deseado

Primero, usamos la opción `-h` para ver el mensaje de ayuda:

```
$ erat -h
Usage:
  erat -h
  erat -V
  erat [options] N
Options:
  -h, --help     Prints usage information.
  -V, --version  Prints version information.
  -o, --output   Path to output file.
Examples:
  erat -o - 10
```

Ahora usaremos el programa para obtener los números primos menores que 10. Usamos "-" como argumento de `-o` para indicarle al programa que imprima el tablero por `stdout`:

```
$ (echo 10) | erat
2
3
5
7
```

El programa deberá retornar un error si su argumento está fuera del rango $[2, \text{MAXINT}]$.

6. Implementación

El programa a implementar deberá satisfacer algunos requerimientos mínimos, que detallamos a continuación.

El propósito de `erat()` es simple: eliminar, de un arreglo de números naturales consecutivos que comienza por 2, todos los números que no sean primos.

```
void erat(unsigned int *p, int n);
```

El programa deberá procesar los argumentos de entrada, crear el arreglo de números naturales consecutivos, de los cuales `erat()` debe poner en 0 los que no sean primos, y escribir en `stdout` o un archivo aquellos que hayan resultado ser primos.

6.1. Algoritmo

El algoritmo a implementar es la Criba de Eratóstenes [3], explicado en clase.

7. Informe

El informe deberá incluir:

- Este enunciado;
- Documentación relevante al diseño e implementación del programa;
- Corridas de prueba para N= -5, 1, 10, 50 y 100, con los comentarios pertinentes;
- El código fuente completo, en dos formatos: digitalizado e impreso en papel.

8. Fecha de entrega

La última fecha de entrega y presentación es el jueves 7 de Septiembre de 2017.

Referencias

[1] GXemul, `http://gavare.se/gxemul/`.

[2] The NetBSD project, `http://www.netbsd.org/`.

[3] Criba de Eratóstenes, `http://http://es.wikipedia.org/wiki/Criba_de_Erat%C3%B3stenes`.