Organización de Computadoras

Trabajo Práctico 1
Implementación en MIPS Assembly

Nicolás Ledesma, *Padrón Nro. 93.118*
nicolas.angel.ledesma@gmail.com

Jonathan Moguilevsky, *Padrón Nro. 95.516*
jmoguilevsky@gmail.com

Leonardo Riego, *Padrón Nro. 94.104*
riegoleonardo@hotmail.com

2do. Cuatrimestre de 2019
66.20 Org. de Computadoras − Práctica: Santi, Pérez Masci, Natale
Facultad de Ingeniería, Universidad de Buenos Aires

Resumen

Este trabajo práctico grupal tiene como objetivo principal.

1. Enunciado

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 1: Programación MIPS
2^{do} cuatrimestre de 2019

`$Date: 2019/10/01 23:05:25 $`

1. Objetivos

Familiarizarse con el conjunto de instrucciones MIPS y el concepto de ABI, extendiendo un programa que resuelva el problema descripto a continuación.

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, un informe impreso de acuerdo con lo que mencionaremos en la sección 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Descripción

El programa, deberá multiplicar matrices cuadradas de números reales, representados en punto flotante de doble precisión.

Las matrices a multiplicar ingresarán como texto por entrada estándar (`stdin`), donde cáda línea describe completamente cada par de matrices a multiplicar, según el siguiente formato:

$N\ a_{1,1}\ a_{1,2}\ ...\ a_{N,N}\ b_{1,1}\ b_{1,2}\ ...\ b_{N,N}$

La línea anterior representa a las matrices A y B, de NxN. Los elementos de la matriz A son los $a_{x,y}$, siendo x e y los indices de fila y columna respectivamente[1]. Los elementos de la matriz B se representan por los $b_{x,y}$ de la misma forma que los de A.

[1] Notar que es una representación del tipo *row major order*, siguiendo el orden en que C dispone las matrices en memoria.

El fin de línea es el caracter \n (*newline*). Los componentes de la línea están separados entre sí por uno o más espacios. El formato de los números en punto flotante son los que corresponden al especificador de conversión 'g' de `printf`[2].

Por ejemplo, dado el siguiente producto de matrices cuadradas:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

Su representación sería:

```
2 1 2 3 4 5 6 7 8
```

Por cada par de matrices que se presenten por cada línea de entrada, el programa deberá multiplicarlas y presentar el resultado por su salida estándar (`stdout`) en el siguiente formato, hasta que llegue al final del archivo de entrada (`EOF`):

$N\ c_{1,1}\ c_{1,2}\ ...\ c_{N,N}$

Ante un error, el progama deberá informar la situación inmediatamente (por `stderr`) y detener su ejecución.

[2] Ver man 3 printf, "Conversion specifiers".

4.1. Ejemplos

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

```
$ tp1 -h
Usage:
  tp1 -h
  tp1 -V
  tp1 < in_file > out_file
Options:
  -V, --version     Print version and quit.
  -h, --help        Print this information and quit.
Examples:
   tp1 < in.txt > out.txt
   cat in.txt | tp1 > out.txt
```

A continuación, ejecutamos algunas pruebas:

```
$ cat example.txt
2 1 2 3 4 1 2 3 4
3 1 2 3 4 5 6.1 3 2 1 1 0 0 0 1 0 0 0 1

$ cat example.txt | ./tp1
2 7 10 15 22
3 1 2 3 4 5 6.1 3 2 1
```

En este ejemplo, realizamos las siguientes multiplicaciones, siendo los miembros izquierdos de la ecuación las matrices de entrada (`stdin`), y los miembros derechos las matrices de salida (`stdout`):

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6.1 \\ 3 & 2 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6.1 \\ 3 & 2 & 1 \end{pmatrix}$$

4.2. Interfaz

Las matrices deberán ser representadas por el tipo de datos `matrix_t`, definido a continuación:

```
typedef struct matrix {
        size_t rows;
        size_t cols;
        double* array;
} matrix_t;
```

Notar que los atributos `rows` y `cols` representan respectivamente la cantidad filas y columnas de la matriz. El atributo `array` contendrá los elementos de la matriz dispuestos en row-major order [3].

Los métodos a implementar, que aplican sobre el tipo de datos `matrix_t` son:

```
// Constructor de matrix_t
matrix_t* create_matrix(size_t rows, size_t cols);

// Destructor de matrix_t
void destroy_matrix(matrix_t* m);

// Imprime matrix_t sobre el file pointer fp en el formato solicitado
// por el enunciado
int print_matrix(FILE* fp, matrix_t* m);

// Multiplica las matrices en m1 y m2
matrix_t* matrix_multiply(matrix_t* m1, matrix_t* m2);
```

5. Implementación

A diferencia del TP anterior, en este caso deberá suministrarse una implementación de `matrix_multiply()` en código assembly MIPS. Así, el resto del programa deberá estar escrito en C, e interactuar con esta función para realiar la multiplicación de matrices.

6. Informe

El informe deberá incluir:

- Documentación relevante al diseño e implementación del programa;
- Comando(s) para compilar el programa;
- Las corridas de prueba, con los comentarios pertinentes;
- El código fuente, en lenguaje C y MIPS;
- El código MIPS32 generado por el compilador[3];

[3] Por motivos prácticos, en la copia impresa sólo es necesario incluir la primera página del código assembly MIPS32 generado por el compilador.

- Este enunciado.

7. Fechas

Fecha de vencimiento: martes 15/10.

Referencias

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

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

[3] Row-major order (Wikipedia), `https://en.wikipedia.org/wiki/Row-major_order`.

2. Solución

2.1. Diseño

La solución propuesta para este trabajo comprende una serie de funciones que pueden ser clasificadas según una determinada división de responsabilidades:

- **Procesamiento:** Comprende las funciones dedicadas a lectura y procesamiento de datos para obtener las matrices a multiplicar.
- **Buffering:** El procesamiento requiere la utilización de un buffer dinámico que almacene los valores numéricos de punto flotante de doble precisión, que primero son almacenados como un array de `char`, y luego convertidos al tipo `double`.
- **Implementación de matrices:** Esta parte comprende la interfaz definida por el enunciado para la creación, impresión, multiplicación y destrucción de matrices.
- **Presentación:** Refiere al código generado para imprimir menús y mensajes de error.

2.1.1. Buffering

Este fragmento de la solución contiene las funciones `init_buffer` y `push`. La primer función inicializa el buffer. La segunda, agrega caracteres a un buffer ingresado como arugmento, manejando dinámicamente la memoria alocada por el buffer on demand.

2.1.2. Procesamiento

Esta parte cuenta con las siguientes funciones:

- `get_value`: Esta función escribe sobre un puntero pasado como argumento el valor de punto flotante obtenido, correspondiente a una 'palabra' leida por `stdin`. Esta misma función se utiliza para leer el primer valor entero que indica el tamaño de la matriz cuadrada. Para esto luego se castea el valor obtenido al tipo `size_t`.

  Para controlar y validar el input, la función retorna la cantidad de caracteres que leyó para obtener el valor.

- `read_matrix`: Aquí se invoca las veces necesarias a la función `get_value` para obtener los valores de cada matriz. En caso de llegar a whitespaces o símbolos no esperados, imprime un error que advierte que el formato del input es inválido. Las matrices las almacena en un puntero a `matrix_t`.

- `process_line`: Como bien lo indica el nombre, esta función procesa una línea de standard input. Lee el primer valor para obtener la dimensión de las matrices. Por cada matriz, invoca a la inicialización de las variables de tipo `matrix_t`, y luego a la función `read_matrix`.

  Por último llama a `matrix_multiply` para multiplicar las matrices y almacenar el resultado en un nuevo puntero a `matrix_t`, para luego imprimir esta última matriz.

2.1.3. Implementación de matrices

Esta sección escencialmente implementa las interfaces propuestas por el enunciado. En particular se destacan las siguiente implementaciones:

1. `create_matrix`: Esta función requiere alocar memoria para la estructura de `matrix_t`, y luego para el array de valores una vez conocido el tamaño de la matriz cuadrada.

   Por ello esta función realiza 2 memory allocation. Finalmente se inicializan los atributos de la matriz correspondientemente.

2. `destroy_matrix`: Como complemento de `create_matrix`, esta función debe liberar la memoria alocada tanto para el array de valores como para la estructura inicializada.

2.1.4. Presentación

Esta sección resuelve la mayor parte de las interacciones con el usuario del programa:

1. `argsHandler_parse_arguments`: Parseo de argumentos y ejecución de la rutina correspondiente.

2. `print_help, print_usage, print_options`:

   Impresión de menú de ayuda y detalle de las opciones de ejecución del programa.

2.1.5. Main function

La función `main` sencillamente invoca al parseo de argumentos para imprimir menús de ayuda u opciones según corresponda. Sino invoca a la rutina que lee el input por `stdin` y procesa las líneas.

2.2. Código

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <getopt.h>

#define VERSION "1.0.0"
#define INIT_SIZE 2

/* ******* Buffer functions ******* */

char *init_buffer(size_t size)
{
    return calloc(size, size * sizeof(char));
}

char *push(char *buffer, char value, size_t *currLength, size_t *
    bufferSize)
```

```
{
    *currLength = *currLength + 1;

    if (*currLength >= *bufferSize)
    {
        char *newBuff;
        *bufferSize = *bufferSize * 2;
        newBuff = realloc(buffer, *bufferSize * sizeof(char));
        newBuff[*currLength - 1] = value;
        return newBuff;
    }
    buffer[*currLength - 1] = value;
    return buffer;
}

/* ******* Matrix implementation ******* */

typedef struct matrix
{
    size_t rows;
    size_t cols;
    double *array;
} matrix_t;

matrix_t *create_matrix(size_t rows, size_t cols)
{
    matrix_t *matrix = malloc(sizeof(matrix_t));
    double *array = malloc(sizeof(double) * cols * rows);
    matrix->array = array;
    matrix->cols = cols;
    matrix->rows = rows;
    return matrix;
}

int print_matrix(FILE *fp, matrix_t *m)
{
    int elementCount = m->rows * m->cols;

    int i = 0;
    for (; i < elementCount; i++)
    {
        char *format = (i == elementCount - 1) ? "%.10g\n" : "%.10g␣";
        int res = fprintf(fp, format, m->array[i]);
        if (res < 0)
        {
            fprintf(stderr, "Error␣writing␣to␣file\n");
            return res;
        }
    }
    return elementCount;
}

matrix_t *matrix_multiply(matrix_t *matrix1, matrix_t *matrix2)
{
```

```
    matrix_t *result = create_matrix(matrix1->rows, matrix2->cols);

    int i = 0;
    for (; i < matrix1->rows; i++)
    {
        int j = 0;
        for (; j < matrix2->cols; j++)
        {
            double acum = 0;
            int k = 0;
            for (; k < matrix1->cols; k++)
            {
                acum = acum + matrix1->array[matrix1->cols * i + k] *
                    matrix2->array[matrix2->cols * k + j];
            }
            result->array[result->cols * i + j] = acum;
        }
    }

    return result;
}

void destroy_matrix(matrix_t *m)
{
    free(m->array);
    free(m);
}

/* ******* Value processing ******* */

int get_value(double *value_ptr, bool *eol, bool *eof)
{
    char c;
    bool stop = false;
    size_t length = 0;
    size_t bufferSize = INIT_SIZE;
    char *buffer = init_buffer(bufferSize);

    while (!stop)
    {
        c = (char)getchar();
        if (c == ' ' || c == '\n' || c == EOF)
        {
            *eol = (c == '\n');
            *eof = (c == EOF);
            stop = true;
        }
        else
        {
            char *new_buffer = push(buffer, c, &length, &bufferSize);
            buffer = new_buffer;
        }
    }
    if (length > 0)
```

```
    {
        buffer[length] = 0;
        int res = sscanf(buffer, "%lG", value_ptr);

        // Si no puede interpretar un double, da error.
        if (res == 0) {
            return -1;
        }
    }
    free(buffer);
    return length;
}

int read_matrix(matrix_t *matrix, int dim, bool *eol, bool *eof, bool
    eolExpected)
{
    int res;
    double value;

    int i = 0;
    for (; i < dim * dim; i++)
    {
        res = get_value(&value, eol, eof);

        if (res < 0 || (res == 0 && (*eol || *eof)))
        {
            fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
            return -1;
        }
        else
        {
            matrix->array[i] = value;
        }

        if (i == (dim * dim - 1) && *eol && !eolExpected) {
            fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
            return -1;
        }
    }

    return 0;
}

enum LineEnding
{
    EndOfLine,
    EndOfFile,
    Error
};

enum LineEnding process_line()
{
    double value;
    int res;
```

```
size_t dim;

bool eol = false;
bool eof = false;

res = get_value(&value, &eol, &eof);

if (eof)
{
    return EndOfFile;
}

if (eol)
{
    fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
    return Error;
}

if (res <= 0)
{
    fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
    return Error;
}

dim = (size_t)value;
int response;

matrix_t *matrix1 = create_matrix(dim, dim);

response = read_matrix(matrix1, dim, &eol, &eof, false);

if (response == -1)
{
    destroy_matrix(matrix1);
    return Error;
}

matrix_t *matrix2 = create_matrix(dim, dim);
response = read_matrix(matrix2, dim, &eol, &eof, true);

if (response == -1)
{
    destroy_matrix(matrix1);
    destroy_matrix(matrix2);
    return Error;
}

if (!eol && !eof)
{
    destroy_matrix(matrix1);
    destroy_matrix(matrix2);
    fprintf(stderr, "Invalid␣format,␣cannot␣read␣matrix\n");
    return Error;
}
```

```
    matrix_t *result = matrix_multiply(matrix1, matrix2);

    printf("%zu␣", dim);
    print_matrix(stdout, result);

    destroy_matrix(matrix1);
    destroy_matrix(matrix2);
    destroy_matrix(result);

    return EndOfLine;
}

static void print_usage(const char *src)
{
    printf("%s\n\t%s%s\n\t%s%s\n\t%s%s\n", "Usage:",
           src, "␣-h",
           src, "␣-V",
           src, "␣<␣in_file␣>␣out_file");
}

static void print_options(const char *src)
{
    printf("%s\n%s\n%s\n%s\n%s%s\t%s\n%s%s%s\n",
           "Options:",
           "\t-V,␣--version\tProgram␣version.",
           "\t-h,␣--help\tPrint␣help.",
           "Examples:",
           "\t", src, "tp0␣<␣in.txt␣>␣out.txt",
           "\tcat␣in.txt␣", src, "␣|␣tp0␣>␣out.txt");
}

static void print_help(const char *src)
{
    print_usage(src);
    print_options(src);
}

static void print_version()
{
    printf("Version␣%s\n", VERSION);
}

int argsHandler_parse_arguments(const int argc, char *const argv[])
{
    int arg;
    int option_index = 0;
    const char *short_opt = "hV";
    static struct option long_options[] = {
       {"help", no_argument, 0, 'h'},
       {"version", no_argument, 0, 'V'},
       {0, 0, 0, 0}};
```

```
    while ((arg = getopt_long(argc, argv, short_opt, long_options, &
        option_index)) != -1)
    {
        switch (arg)
        {
        case 'h':
            print_help(argv[0]);
            exit(EXIT_SUCCESS);
        case 'V':
            print_version();
            exit(EXIT_SUCCESS);
        case '?':
            exit(EXIT_FAILURE);
            break;
        default:
            break;
        }
    }

    // extern int optind
    if (optind < argc)
    {
        fprintf(stderr, "Unrecognized␣options:␣");
        while (optind < argc)
        {
            fprintf(stderr, "%s␣", argv[optind++]);
        }
        fprintf(stderr, "\n");
        exit(EXIT_FAILURE);
    }

    return EXIT_SUCCESS;
}

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

    enum LineEnding result;

    do
    {
        result = process_line();
    } while (result == EndOfLine && result != Error);

    if (result == Error)
    {
        return -1;
    }

    return 0;
}
```

2.3. Compilación del programa

Para compilar el programa se utiliza el makefile desde el directorio donde se descomprimió el entregable:

```
$ make clean
$ make all
gcc -std=c99 -Wall -Werror -pedantic -pedantic-errors -c dynamic.c
gcc -o dynamic dynamic.o
```

3. Pruebas

3.0.1. Ejecución

Para la ejecución de las pruebas generamos un script llamado `run_tests.sh` que se encarga de, para cada caso generado, ejecutar el programa y escribit el output en un archivo. Finalmente, se comparan los outputs contra archivos que contienen los resultados esperados, en la carpeta `results/`.

3.1. Casos de prueba

Para probar nuestro programa generamos una serie de casos de prueba:

- `big_matrix_ints.txt`: Matriz de 100x100 con valores enteros
- `big_matrix_doubles.txt`: Matriz de 100x100 con valores de punto flotante
- `exponential_doubles_matrix.txt`: Matriz pequeña con valores de punto flotante en notación exponencial.
- `invalid_input.txt`: Prueba con input invalido.
- `simple_multiline.txt`: Prueba de input con varias líneas.
- `single_line.txt`: Prueba simple de una sola línea.
- `small_double_matrix.txt`: Prueba simple de una matriz 3x3 de doubles.
- `small_int_matrix.txt`: Prueba simple de una matriz 3x3 de integers.

La ejecución de los casos se realiza de la siguiente manera:

```
$ ./run_tests.sh
```

Este script de ejecución valida también la impresión del menú y de la versión del programa.

4. Código MIPS

4.1. Código assembly generado para *dynamic.c*

```
  .file 1 "dynamic.c"
  .section .mdebug.abi32
  .previous
  .abicalls
  .text
  .align 2
  .globl init_buffer
  .ent init_buffer
init_buffer:
  .frame $fp,40,$ra  # vars= 0, regs= 3/0, args= 16, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,40
  .cprestore 16
  sw $ra,32($sp)
  sw $fp,28($sp)
  sw $gp,24($sp)
  move $fp,$sp
  sw $a0,40($fp)
  lw $a0,40($fp)
  lw $a1,40($fp)
  la $t9,calloc
  jal $ra,$t9
  move $sp,$fp
  lw $ra,32($sp)
  lw $fp,28($sp)
  addu $sp,$sp,40
  j $ra
  .end init_buffer
  .size init_buffer, .-init_buffer
  .align 2
  .globl push
  .ent push
push:
  .frame $fp,56,$ra  # vars= 16, regs= 3/0, args= 16, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,56
  .cprestore 16
  sw $ra,48($sp)
  sw $fp,44($sp)
  sw $gp,40($sp)
  move $fp,$sp
  sw $a0,56($fp)
  move $v0,$a1
  sw $a2,64($fp)
  sw $a3,68($fp)
```

```
  sb $v0,24($fp)
  lw $v1,64($fp)
  lw $v0,64($fp)
  lw $v0,0($v0)
  addu $v0,$v0,1
  sw $v0,0($v1)
  lw $v0,64($fp)
  lw $v1,68($fp)
  lw $a0,0($v0)
  lw $v0,0($v1)
  sltu $v0,$a0,$v0
  bne $v0,$zero,$L19
  lw $v1,68($fp)
  lw $v0,68($fp)
  lw $v0,0($v0)
  sll $v0,$v0,1
  sw $v0,0($v1)
  lw $v0,68($fp)
  lw $a0,56($fp)
  lw $a1,0($v0)
  la $t9,realloc
  jal $ra,$t9
  sw $v0,28($fp)
  lw $v0,64($fp)
  lw $v1,28($fp)
  lw $v0,0($v0)
  addu $v0,$v1,$v0
  addu $v1,$v0,-1
  lbu $v0,24($fp)
  sb $v0,0($v1)
  lw $v0,28($fp)
  sw $v0,32($fp)
  b $L18
$L19:
  lw $v0,64($fp)
  lw $v1,56($fp)
  lw $v0,0($v0)
  addu $v0,$v1,$v0
  addu $v1,$v0,-1
  lbu $v0,24($fp)
  sb $v0,0($v1)
  lw $v0,56($fp)
  sw $v0,32($fp)
$L18:
  lw $v0,32($fp)
  move $sp,$fp
  lw $ra,48($sp)
  lw $fp,44($sp)
  addu $sp,$sp,56
  j $ra
  .end push
  .size push, .-push
  .align 2
  .globl create_matrix
```

```
  .ent create_matrix
create_matrix:
  .frame $fp,48,$ra  # vars= 8, regs= 3/0, args= 16, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,48
  .cprestore 16
  sw $ra,40($sp)
  sw $fp,36($sp)
  sw $gp,32($sp)
  move $fp,$sp
  sw $a0,48($fp)
  sw $a1,52($fp)
  li $a0,12   # 0xc
  la $t9,malloc
  jal $ra,$t9
  sw $v0,24($fp)
  lw $v1,52($fp)
  lw $v0,48($fp)
  mult $v1,$v0
  mflo $v0
  sll $v0,$v0,3
  move $a0,$v0
  la $t9,malloc
  jal $ra,$t9
  sw $v0,28($fp)
  lw $v1,24($fp)
  lw $v0,28($fp)
  sw $v0,8($v1)
  lw $v1,24($fp)
  lw $v0,52($fp)
  sw $v0,4($v1)
  lw $v1,24($fp)
  lw $v0,48($fp)
  sw $v0,0($v1)
  lw $v0,24($fp)
  move $sp,$fp
  lw $ra,40($sp)
  lw $fp,36($sp)
  addu $sp,$sp,48
  j $ra
  .end create_matrix
  .size create_matrix, .-create_matrix
  .rdata
  .align 2
$LC0:
  .ascii "%.10g\n\000"
  .align 2
$LC1:
  .ascii "%.10g␣\000"
  .align 2
```

```
$LC2:
  .ascii "Error␣writing␣to␣file\n\000"
  .text
  .align 2
  .globl print_matrix
  .ent print_matrix
print_matrix:
  .frame $fp,64,$ra  # vars= 24, regs= 3/0, args= 16, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,64
  .cprestore 16
  sw $ra,56($sp)
  sw $fp,52($sp)
  sw $gp,48($sp)
  move $fp,$sp
  sw $a0,64($fp)
  sw $a1,68($fp)
  lw $v0,68($fp)
  lw $v1,68($fp)
  lw $a0,0($v0)
  lw $v0,4($v1)
  mult $a0,$v0
  mflo $v0
  sw $v0,24($fp)
  sw $zero,28($fp)
$L22:
  lw $v0,28($fp)
  lw $v1,24($fp)
  slt $v0,$v0,$v1
  bne $v0,$zero,$L25
  b $L23
$L25:
  lw $v0,24($fp)
  addu $v1,$v0,-1
  lw $v0,28($fp)
  bne $v0,$v1,$L26
  la $v0,$LC0
  sw $v0,44($fp)
  b $L27
$L26:
  la $v0,$LC1
  sw $v0,44($fp)
$L27:
  lw $v0,44($fp)
  sw $v0,32($fp)
  lw $a0,68($fp)
  lw $v0,28($fp)
  sll $v1,$v0,3
  lw $v0,8($a0)
  addu $v0,$v1,$v0
```

```
  lw $a0,64($fp)
  lw $a1,32($fp)
  lw $a2,0($v0)
  lw $a3,4($v0)
  la $t9,fprintf
  jal $ra,$t9
  sw $v0,36($fp)
  lw $v0,36($fp)
  bgez $v0,$L24
  la $a0,__sF+176
  la $a1,$LC2
  la $t9,fprintf
  jal $ra,$t9
  lw $v0,36($fp)
  sw $v0,40($fp)
  b $L21
$L24:
  lw $v0,28($fp)
  addu $v0,$v0,1
  sw $v0,28($fp)
  b $L22
$L23:
  lw $v0,24($fp)
  sw $v0,40($fp)
$L21:
  lw $v0,40($fp)
  move $sp,$fp
  lw $ra,56($sp)
  lw $fp,52($sp)
  addu $sp,$sp,64
  j $ra
  .end print_matrix
  .size print_matrix, .-print_matrix
  .align 2
  .globl matrix_multiply
  .ent matrix_multiply
matrix_multiply:
  .frame $fp,72,$ra  # vars= 32, regs= 3/0, args= 16, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,72
  .cprestore 16
  sw $ra,64($sp)
  sw $fp,60($sp)
  sw $gp,56($sp)
  move $fp,$sp
  sw $a0,72($fp)
  sw $a1,76($fp)
  lw $v0,72($fp)
  lw $v1,76($fp)
  lw $a0,0($v0)
```

```
  lw $a1,4($v1)
  la $t9,create_matrix
  jal $ra,$t9
  sw $v0,24($fp)
  sw $zero,28($fp)
$L30:
  lw $v0,72($fp)
  lw $v1,28($fp)
  lw $v0,0($v0)
  sltu $v0,$v1,$v0
  bne $v0,$zero,$L33
  b $L31
$L33:
  sw $zero,32($fp)
$L34:
  lw $v0,76($fp)
  lw $v1,32($fp)
  lw $v0,4($v0)
  sltu $v0,$v1,$v0
  bne $v0,$zero,$L37
  b $L32
$L37:
  sw $zero,40($fp)
  sw $zero,44($fp)
  sw $zero,48($fp)
$L38:
  lw $v0,72($fp)
  lw $v1,48($fp)
  lw $v0,4($v0)
  sltu $v0,$v1,$v0
  bne $v0,$zero,$L41
  b $L39
$L41:
  lw $a0,72($fp)
  lw $v0,72($fp)
  lw $v1,4($v0)
  lw $v0,28($fp)
  mult $v1,$v0
  mflo $v1
  lw $v0,48($fp)
  addu $v0,$v1,$v0
  sll $v1,$v0,3
  lw $v0,8($a0)
  addu $a1,$v1,$v0
  lw $a0,76($fp)
  lw $v0,76($fp)
  lw $v1,4($v0)
  lw $v0,48($fp)
  mult $v1,$v0
  mflo $v1
  lw $v0,32($fp)
  addu $v0,$v1,$v0
  sll $v1,$v0,3
  lw $v0,8($a0)
```

```
  addu $v0,$v1,$v0
  l.d $f2,0($a1)
  l.d $f0,0($v0)
  mul.d $f2,$f2,$f0
  l.d $f0,40($fp)
  add.d $f0,$f0,$f2
  s.d $f0,40($fp)
  lw $v0,48($fp)
  addu $v0,$v0,1
  sw $v0,48($fp)
  b $L38
$L39:
  lw $a0,24($fp)
  lw $v0,24($fp)
  lw $v1,4($v0)
  lw $v0,28($fp)
  mult $v1,$v0
  mflo $v1
  lw $v0,32($fp)
  addu $v0,$v1,$v0
  sll $v1,$v0,3
  lw $v0,8($a0)
  addu $v0,$v1,$v0
  l.d $f0,40($fp)
  s.d $f0,0($v0)
  lw $v0,32($fp)
  addu $v0,$v0,1
  sw $v0,32($fp)
  b $L34
$L32:
  lw $v0,28($fp)
  addu $v0,$v0,1
  sw $v0,28($fp)
  b $L30
$L31:
  lw $v0,24($fp)
  move $sp,$fp
  lw $ra,64($sp)
  lw $fp,60($sp)
  addu $sp,$sp,72
  j $ra
  .end matrix_multiply
  .size matrix_multiply, .-matrix_multiply
  .align 2
  .globl destroy_matrix
  .ent destroy_matrix
destroy_matrix:
  .frame $fp,40,$ra  # vars= 0, regs= 3/0, args= 16, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,40
```

```
  .cprestore 16
  sw $ra,32($sp)
  sw $fp,28($sp)
  sw $gp,24($sp)
  move $fp,$sp
  sw $a0,40($fp)
  lw $v0,40($fp)
  lw $a0,8($v0)
  la $t9,free
  jal $ra,$t9
  lw $a0,40($fp)
  la $t9,free
  jal $ra,$t9
  move $sp,$fp
  lw $ra,32($sp)
  lw $fp,28($sp)
  addu $sp,$sp,40
  j $ra
  .end destroy_matrix
  .size destroy_matrix, .-destroy_matrix
  .rdata
  .align 2
$LC3:
  .ascii "%lG\000"
  .text
  .align 2
  .globl get_value
  .ent get_value
get_value:
  .frame $fp,72,$ra  # vars= 32, regs= 3/0, args= 16, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,72
  .cprestore 16
  sw $ra,64($sp)
  sw $fp,60($sp)
  sw $gp,56($sp)
  move $fp,$sp
  sw $a0,72($fp)
  sw $a1,76($fp)
  sw $a2,80($fp)
  sb $zero,25($fp)
  sw $zero,28($fp)
  li $v0,2   # 0x2
  sw $v0,32($fp)
  lw $a0,32($fp)
  la $t9,init_buffer
  jal $ra,$t9
  sw $v0,36($fp)
$L44:
  lbu $v0,25($fp)
```

```
  beq $v0,$zero,$L46
  b $L45
$L46:
  lw $v0,__sF+4
  addu $v0,$v0,-1
  sw $v0,__sF+4
  bgez $v0,$L47
  la $a0,__sF
  la $t9,__srget
  jal $ra,$t9
  sb $v0,48($fp)
  b $L48
$L47:
  la $v0,__sF
  lw $v1,0($v0)
  move $a0,$v1
  lbu $a0,0($a0)
  sb $a0,48($fp)
  addu $v1,$v1,1
  sw $v1,0($v0)
$L48:
  lbu $v0,48($fp)
  sb $v0,24($fp)
  lb $v1,24($fp)
  li $v0,32   # 0x20
  beq $v1,$v0,$L50
  lb $v1,24($fp)
  li $v0,10   # 0xa
  beq $v1,$v0,$L50
  lb $v1,24($fp)
  li $v0,-1   # 0xffffffffffffffff
  beq $v1,$v0,$L50
  b $L49
$L50:
  lw $v1,76($fp)
  lb $v0,24($fp)
  xori $v0,$v0,0xa
  sltu $v0,$v0,1
  sb $v0,0($v1)
  lw $a0,80($fp)
  lb $v1,24($fp)
  li $v0,-1   # 0xffffffffffffffff
  xor $v0,$v1,$v0
  sltu $v0,$v0,1
  sb $v0,0($a0)
  li $v0,1   # 0x1
  sb $v0,25($fp)
  b $L44
$L49:
  lb $v0,24($fp)
  addu $v1,$fp,28
  addu $a3,$fp,32
  lw $a0,36($fp)
  move $a1,$v0
```

```
  move $a2,$v1
  la $t9,push
  jal $ra,$t9
  sw $v0,40($fp)
  lw $v0,40($fp)
  sw $v0,36($fp)
  b $L44
$L45:
  lw $v0,28($fp)
  beq $v0,$zero,$L52
  lw $v1,36($fp)
  lw $v0,28($fp)
  addu $v0,$v1,$v0
  sb $zero,0($v0)
  lw $a0,36($fp)
  la $a1,$LC3
  lw $a2,72($fp)
  la $t9,sscanf
  jal $ra,$t9
  sw $v0,40($fp)
  lw $v0,40($fp)
  bne $v0,$zero,$L52
  li $v0,-1   # 0xffffffffffffffff
  sw $v0,44($fp)
  b $L43
$L52:
  lw $a0,36($fp)
  la $t9,free
  jal $ra,$t9
  lw $v0,28($fp)
  sw $v0,44($fp)
$L43:
  lw $v0,44($fp)
  move $sp,$fp
  lw $ra,64($sp)
  lw $fp,60($sp)
  addu $sp,$sp,72
  j $ra
  .end get_value
  .size get_value, .-get_value
  .rdata
  .align 2
$LC4:
  .ascii "Invalid␣format,␣cannot␣read␣matrix\n\000"
  .text
  .align 2
  .globl read_matrix
  .ent read_matrix
read_matrix:
  .frame $fp,64,$ra  # vars= 24, regs= 3/0, args= 16, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
```

```
  .set reorder
  subu $sp,$sp,64
  .cprestore 16
  sw $ra,56($sp)
  sw $fp,52($sp)
  sw $gp,48($sp)
  move $fp,$sp
  sw $a0,64($fp)
  sw $a1,68($fp)
  sw $a2,72($fp)
  sw $a3,76($fp)
  lw $v0,80($fp)
  sb $v0,24($fp)
  sw $zero,40($fp)
$L55:
  lw $v1,68($fp)
  lw $v0,68($fp)
  mult $v1,$v0
  mflo $v1
  lw $v0,40($fp)
  slt $v0,$v0,$v1
  bne $v0,$zero,$L58
  b $L56
$L58:
  addu $v0,$fp,32
  move $a0,$v0
  lw $a1,72($fp)
  lw $a2,76($fp)
  la $t9,get_value
  jal $ra,$t9
  sw $v0,28($fp)
  lw $v0,28($fp)
  bltz $v0,$L60
  lw $v0,28($fp)
  bne $v0,$zero,$L59
  lw $v0,72($fp)
  lbu $v0,0($v0)
  bne $v0,$zero,$L60
  lw $v0,76($fp)
  lbu $v0,0($v0)
  bne $v0,$zero,$L60
  b $L59
$L60:
  la $a0,__sF+176
  la $a1,$LC4
  la $t9,fprintf
  jal $ra,$t9
  li $v0,-1   # 0xffffffffffffffff
  sw $v0,44($fp)
  b $L54
$L59:
  lw $a0,64($fp)
  lw $v0,40($fp)
  sll $v1,$v0,3
```

```
  lw $v0,8($a0)
  addu $v0,$v1,$v0
  l.d $f0,32($fp)
  s.d $f0,0($v0)
  lw $v1,68($fp)
  lw $v0,68($fp)
  mult $v1,$v0
  mflo $v0
  addu $v1,$v0,-1
  lw $v0,40($fp)
  bne $v0,$v1,$L57
  lw $v0,72($fp)
  lbu $v0,0($v0)
  beq $v0,$zero,$L57
  lbu $v0,24($fp)
  bne $v0,$zero,$L57
  la $a0,__sF+176
  la $a1,$LC4
  la $t9,fprintf
  jal $ra,$t9
  li $v0,-1   # 0xffffffffffffffff
  sw $v0,44($fp)
  b $L54
$L57:
  lw $v0,40($fp)
  addu $v0,$v0,1
  sw $v0,40($fp)
  b $L55
$L56:
  sw $zero,44($fp)
$L54:
  lw $v0,44($fp)
  move $sp,$fp
  lw $ra,56($sp)
  lw $fp,52($sp)
  addu $sp,$sp,64
  j $ra
  .end read_matrix
  .size read_matrix, .-read_matrix
  .rdata
  .align 2
$LC6:
  .ascii "%zu␣\000"
  .align 3
$LC5:
  .word 0
  .word 1105199104
  .text
  .align 2
  .globl process_line
  .ent process_line
process_line:
  .frame $fp,112,$ra  # vars= 64, regs= 3/0, args= 24, extra= 8
  .mask 0xd0000000,-8
```

```
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,112
  .cprestore 24
  sw $ra,104($sp)
  sw $fp,100($sp)
  sw $gp,96($sp)
  move $fp,$sp
  sb $zero,48($fp)
  sb $zero,49($fp)
  addu $v0,$fp,48
  addu $v1,$fp,49
  addu $a0,$fp,32
  move $a1,$v0
  move $a2,$v1
  la $t9,get_value
  jal $ra,$t9
  sw $v0,40($fp)
  lbu $v0,49($fp)
  beq $v0,$zero,$L64
  li $v0,1   # 0x1
  sw $v0,68($fp)
  b $L63
$L64:
  lbu $v0,48($fp)
  beq $v0,$zero,$L65
  la $a0,__sF+176
  la $a1,$LC4
  la $t9,fprintf
  jal $ra,$t9
  li $v1,2   # 0x2
  sw $v1,68($fp)
  b $L63
$L65:
  lw $v0,40($fp)
  bgtz $v0,$L66
  la $a0,__sF+176
  la $a1,$LC4
  la $t9,fprintf
  jal $ra,$t9
  li $a0,2   # 0x2
  sw $a0,68($fp)
  b $L63
$L66:
  l.d $f0,32($fp)
  s.d $f0,72($fp)
  l.d $f2,$LC5
  s.d $f2,88($fp)
  l.d $f4,72($fp)
  l.d $f0,88($fp)
  c.le.d $f0,$f4
  bc1t $L67
```

```
  l.d $f0,72($fp)
  trunc.w.d $f0,$f0,$v0
  s.s $f0,80($fp)
  b $L68
$L67:
  l.d $f2,72($fp)
  l.d $f4,88($fp)
  sub.d $f0,$f2,$f4
  li $v0,-2147483648   # 0xffffffff80000000
  trunc.w.d $f2,$f0,$a0
  s.s $f2,80($fp)
  lw $v1,80($fp)
  or $v1,$v1,$v0
  sw $v1,80($fp)
$L68:
  lw $a0,80($fp)
  sw $a0,44($fp)
  lw $a0,44($fp)
  lw $a1,44($fp)
  la $t9,create_matrix
  jal $ra,$t9
  sw $v0,56($fp)
  addu $v0,$fp,48
  addu $v1,$fp,49
  sw $zero,16($sp)
  lw $a0,56($fp)
  lw $a1,44($fp)
  move $a2,$v0
  move $a3,$v1
  la $t9,read_matrix
  jal $ra,$t9
  sw $v0,52($fp)
  lw $v1,52($fp)
  li $v0,-1   # 0xffffffffffffffff
  bne $v1,$v0,$L69
  lw $a0,56($fp)
  la $t9,destroy_matrix
  jal $ra,$t9
  li $a1,2   # 0x2
  sw $a1,68($fp)
  b $L63
$L69:
  lw $a0,44($fp)
  lw $a1,44($fp)
  la $t9,create_matrix
  jal $ra,$t9
  sw $v0,60($fp)
  addu $v1,$fp,48
  addu $a3,$fp,49
  li $v0,1   # 0x1
  sw $v0,16($sp)
  lw $a0,60($fp)
  lw $a1,44($fp)
  move $a2,$v1
```

```
  la $t9,read_matrix
  jal $ra,$t9
  sw $v0,52($fp)
  lw $v1,52($fp)
  li $v0,-1   # 0xffffffffffffffff
  bne $v1,$v0,$L70
  lw $a0,56($fp)
  la $t9,destroy_matrix
  jal $ra,$t9
  lw $a0,60($fp)
  la $t9,destroy_matrix
  jal $ra,$t9
  li $v0,2   # 0x2
  sw $v0,68($fp)
  b $L63
$L70:
  lbu $v0,48($fp)
  bne $v0,$zero,$L71
  lbu $v0,49($fp)
  bne $v0,$zero,$L71
  lw $a0,56($fp)
  la $t9,destroy_matrix
  jal $ra,$t9
  lw $a0,60($fp)
  la $t9,destroy_matrix
  jal $ra,$t9
  la $a0,__sF+176
  la $a1,$LC4
  la $t9,fprintf
  jal $ra,$t9
  li $v1,2   # 0x2
  sw $v1,68($fp)
  b $L63
$L71:
  lw $a0,56($fp)
  lw $a1,60($fp)
  la $t9,matrix_multiply
  jal $ra,$t9
  sw $v0,64($fp)
  la $a0,$LC6
  lw $a1,44($fp)
  la $t9,printf
  jal $ra,$t9
  la $a0,__sF+88
  lw $a1,64($fp)
  la $t9,print_matrix
  jal $ra,$t9
  lw $a0,56($fp)
  la $t9,destroy_matrix
  jal $ra,$t9
  lw $a0,60($fp)
  la $t9,destroy_matrix
  jal $ra,$t9
  lw $a0,64($fp)
```

```
  la $t9,destroy_matrix
  jal $ra,$t9
  sw $zero,68($fp)
$L63:
  lw $v0,68($fp)
  move $sp,$fp
  lw $ra,104($sp)
  lw $fp,100($sp)
  addu $sp,$sp,112
  j $ra
  .end process_line
  .size process_line, .-process_line
  .rdata
  .align 2
$LC10:
  .ascii "␣-V\000"
  .align 2
$LC11:
  .ascii "␣<␣in_file␣>␣out_file\000"
  .align 2
$LC7:
  .ascii "%s\n"
  .ascii "\t%s%s\n"
  .ascii "\t%s%s\n"
  .ascii "\t%s%s\n\000"
  .align 2
$LC8:
  .ascii "Usage:\000"
  .align 2
$LC9:
  .ascii "␣-h\000"
  .text
  .align 2
  .ent print_usage
print_usage:
  .frame $fp,56,$ra  # vars= 0, regs= 3/0, args= 32, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,56
  .cprestore 32
  sw $ra,48($sp)
  sw $fp,44($sp)
  sw $gp,40($sp)
  move $fp,$sp
  sw $a0,56($fp)
  lw $v0,56($fp)
  sw $v0,16($sp)
  la $v0,$LC10
  sw $v0,20($sp)
  lw $v0,56($fp)
  sw $v0,24($sp)
```

```
  la $v0,$LC11
  sw $v0,28($sp)
  la $a0,$LC7
  la $a1,$LC8
  lw $a2,56($fp)
  la $a3,$LC9
  la $t9,printf
  jal $ra,$t9
  move $sp,$fp
  lw $ra,48($sp)
  lw $fp,44($sp)
  addu $sp,$sp,56
  j $ra
  .end print_usage
  .size print_usage, .-print_usage
  .rdata
  .align 2
$LC16:
  .ascii "Examples:\000"
  .align 2
$LC17:
  .ascii "\t\000"
  .align 2
$LC18:
  .ascii "tp0␣<␣in.txt␣>␣out.txt\000"
  .align 2
$LC19:
  .ascii "\tcat␣in.txt␣\000"
  .align 2
$LC20:
  .ascii "␣|␣tp0␣>␣out.txt\000"
  .align 2
$LC12:
  .ascii "%s\n"
  .ascii "%s\n"
  .ascii "%s\n"
  .ascii "%s\n"
  .ascii "%s%s\t%s\n"
  .ascii "%s%s%s\n\000"
  .align 2
$LC13:
  .ascii "Options:\000"
  .align 2
$LC14:
  .ascii "\t-V,␣--version\tProgram␣version.\000"
  .align 2
$LC15:
  .ascii "\t-h,␣--help\tPrint␣help.\000"
  .text
  .align 2
  .ent print_options
print_options:
  .frame $fp,72,$ra  # vars= 0, regs= 3/0, args= 48, extra= 8
  .mask 0xd0000000,-8
```

```
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,72
  .cprestore 48
  sw $ra,64($sp)
  sw $fp,60($sp)
  sw $gp,56($sp)
  move $fp,$sp
  sw $a0,72($fp)
  la $v0,$LC16
  sw $v0,16($sp)
  la $v0,$LC17
  sw $v0,20($sp)
  lw $v0,72($fp)
  sw $v0,24($sp)
  la $v0,$LC18
  sw $v0,28($sp)
  la $v0,$LC19
  sw $v0,32($sp)
  lw $v0,72($fp)
  sw $v0,36($sp)
  la $v0,$LC20
  sw $v0,40($sp)
  la $a0,$LC12
  la $a1,$LC13
  la $a2,$LC14
  la $a3,$LC15
  la $t9,printf
  jal $ra,$t9
  move $sp,$fp
  lw $ra,64($sp)
  lw $fp,60($sp)
  addu $sp,$sp,72
  j $ra
  .end print_options
  .size print_options, .-print_options
  .align 2
  .ent print_help
print_help:
  .frame $fp,40,$ra  # vars= 0, regs= 3/0, args= 16, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,40
  .cprestore 16
  sw $ra,32($sp)
  sw $fp,28($sp)
  sw $gp,24($sp)
  move $fp,$sp
  sw $a0,40($fp)
```

```
  lw $a0,40($fp)
  la $t9,print_usage
  jal $ra,$t9
  lw $a0,40($fp)
  la $t9,print_options
  jal $ra,$t9
  move $sp,$fp
  lw $ra,32($sp)
  lw $fp,28($sp)
  addu $sp,$sp,40
  j $ra
  .end print_help
  .size print_help, .-print_help
  .rdata
  .align 2
$LC21:
  .ascii "Version␣%s\n\000"
  .align 2
$LC22:
  .ascii "1.0.0\000"
  .text
  .align 2
  .ent print_version
print_version:
  .frame $fp,40,$ra  # vars= 0, regs= 3/0, args= 16, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,40
  .cprestore 16
  sw $ra,32($sp)
  sw $fp,28($sp)
  sw $gp,24($sp)
  move $fp,$sp
  la $a0,$LC21
  la $a1,$LC22
  la $t9,printf
  jal $ra,$t9
  move $sp,$fp
  lw $ra,32($sp)
  lw $fp,28($sp)
  addu $sp,$sp,40
  j $ra
  .end print_version
  .size print_version, .-print_version
  .rdata
  .align 2
$LC24:
  .ascii "help\000"
  .align 2
$LC25:
  .ascii "version\000"
```

```
  .data
  .align 2
  .type long_options.0, @object
  .size long_options.0, 48
long_options.0:
  .word $LC24
  .word 0
  .word 0
  .word 104
  .word $LC25
  .word 0
  .word 0
  .word 86
  .word 0
  .word 0
  .word 0
  .word 0
  .rdata
  .align 2
$LC23:
  .ascii "hV\000"
  .align 2
$LC26:
  .ascii "Unrecognized␣options:␣\000"
  .align 2
$LC27:
  .ascii "%s␣\000"
  .align 2
$LC28:
  .ascii "\n\000"
  .text
  .align 2
  .globl argsHandler_parse_arguments
  .ent argsHandler_parse_arguments
argsHandler_parse_arguments:
  .frame $fp,64,$ra  # vars= 16, regs= 3/0, args= 24, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,64
  .cprestore 24
  sw $ra,56($sp)
  sw $fp,52($sp)
  sw $gp,48($sp)
  move $fp,$sp
  sw $a0,64($fp)
  sw $a1,68($fp)
  sw $zero,36($fp)
  la $v0,$LC23
  sw $v0,40($fp)
$L77:
  addu $v0,$fp,36
```

```
  sw $v0,16($sp)
  lw $a0,64($fp)
  lw $a1,68($fp)
  lw $a2,40($fp)
  la $a3,long_options.0
  la $t9,getopt_long
  jal $ra,$t9
  sw $v0,32($fp)
  lw $v1,32($fp)
  li $v0,-1   # 0xffffffffffffffff
  bne $v1,$v0,$L79
  b $L78
$L79:
  lw $v0,32($fp)
  sw $v0,44($fp)
  li $v0,86   # 0x56
  lw $v1,44($fp)
  beq $v1,$v0,$L82
  lw $v1,44($fp)
  slt $v0,$v1,87
  beq $v0,$zero,$L86
  li $v0,63   # 0x3f
  lw $v1,44($fp)
  beq $v1,$v0,$L83
  b $L77
$L86:
  li $v0,104   # 0x68
  lw $v1,44($fp)
  beq $v1,$v0,$L81
  b $L77
$L81:
  lw $v0,68($fp)
  lw $a0,0($v0)
  la $t9,print_help
  jal $ra,$t9
  move $a0,$zero
  la $t9,exit
  jal $ra,$t9
$L82:
  la $t9,print_version
  jal $ra,$t9
  move $a0,$zero
  la $t9,exit
  jal $ra,$t9
$L83:
  li $a0,1   # 0x1
  la $t9,exit
  jal $ra,$t9
$L78:
  lw $v0,optind
  lw $v1,64($fp)
  slt $v0,$v0,$v1
  beq $v0,$zero,$L87
  la $a0,__sF+176
```

```
  la $a1,$LC26
  la $t9,fprintf
  jal $ra,$t9
$L88:
  lw $v0,optind
  lw $v1,64($fp)
  slt $v0,$v0,$v1
  bne $v0,$zero,$L90
  b $L89
$L90:
  la $a1,optind
  lw $v1,0($a1)
  move $v0,$v1
  sll $a0,$v0,2
  lw $v0,68($fp)
  addu $v0,$a0,$v0
  addu $v1,$v1,1
  sw $v1,0($a1)
  la $a0,__sF+176
  la $a1,$LC27
  lw $a2,0($v0)
  la $t9,fprintf
  jal $ra,$t9
  b $L88
$L89:
  la $a0,__sF+176
  la $a1,$LC28
  la $t9,fprintf
  jal $ra,$t9
  li $a0,1   # 0x1
  la $t9,exit
  jal $ra,$t9
$L87:
  move $v0,$zero
  move $sp,$fp
  lw $ra,56($sp)
  lw $fp,52($sp)
  addu $sp,$sp,64
  j $ra
  .end argsHandler_parse_arguments
  .size argsHandler_parse_arguments, .-argsHandler_parse_arguments
  .align 2
  .globl main
  .ent main
main:
  .frame $fp,48,$ra  # vars= 8, regs= 3/0, args= 16, extra= 8
  .mask 0xd0000000,-8
  .fmask 0x00000000,0
  .set noreorder
  .cpload $t9
  .set reorder
  subu $sp,$sp,48
  .cprestore 16
  sw $ra,40($sp)
```

```
  sw $fp,36($sp)
  sw $gp,32($sp)
  move $fp,$sp
  sw $a0,48($fp)
  sw $a1,52($fp)
  lw $a0,48($fp)
  lw $a1,52($fp)
  la $t9,argsHandler_parse_arguments
  jal $ra,$t9
$L92:
  la $t9,process_line
  jal $ra,$t9
  sw $v0,24($fp)
  lw $v0,24($fp)
  bne $v0,$zero,$L93
  lw $v1,24($fp)
  li $v0,2   # 0x2
  bne $v1,$v0,$L92
$L93:
  lw $v1,24($fp)
  li $v0,2   # 0x2
  bne $v1,$v0,$L97
  li $v0,-1   # 0xffffffffffffffff
  sw $v0,28($fp)
  b $L91
$L97:
  sw $zero,28($fp)
$L91:
  lw $v0,28($fp)
  move $sp,$fp
  lw $ra,40($sp)
  lw $fp,36($sp)
  addu $sp,$sp,48
  j $ra
  .end main
  .size main, .-main
  .ident "GCC:␣(GNU)␣3.3.3␣(NetBSD␣nb3␣20040520)"
```

5. Conclusiones

Como conclusión podemos enumerar una serie de conocimientos que adquirimos y/o pusimos en práctica durante el desarrollo de este trabajo práctico.

A modo de conclusión podemos decir que en este trabajo práctico pudimos configurar y familiarizarnos con el entorno de trabajo de la materia, que emula una computadora con procesador MIPS. . Además adquirimos conocimiento respecto a otras herramientas útiles como ssh, scp, y diff entre otras.

- Configurar y familiarizarnos con el entorno de trabajo de la materia, que emula una computadora con procesador MIPS.
- También pudimos observar algunas de las diferencias entre programar para un sistema operativo linux y para el netBSD que corre emulado.

- Empleamos comandos y herramientas útiles como ssh, scp, diff, echo, cat, entre otros.
- Implementamos un buffer de manejo dinámico de memoria.
- Utilizamos valgrind para analizar la memoria alocada durante la ejecución, detectar leaks tanto de heap como de stack, y revisar los accesos a memoria con punteros.