# 6620 - Organización de Computadoras

Beltrán, Belén (91718)

Forlenza, Marcos (87237)

**Resumen**

El presente trabajo tiene como objetivo la familiarización con las herramientas de software que se utilizarán a lo largo de los siguientes trabajos prácticos, por lo que se lleva a cabo la implementación de un programa que resuelve cierta problemática detallada en los próximos apartados.

# Índice

# 1. Introducción

El programa realizado consta de un computador de autómatas celulares para cierta regla arbitraria. Devuelve como resultado un archivo .pbm con la evolución de este autómata celular para cierta regla determinada.

La implementación del programa se realizará en el lenguaje de programación C. Luego se ejecutará la aplicación sobre una plataforma *NetBSD/MIPS-32* mediante el emulador *GXEmul* [1].

# 2. Compilación

La herramienta para compilar el código en lenguaje C será el *GCC*.

Para automatizar las tareas de compilación se hace uso de la herramienta *GNU Make*. Los Makefiles utilizados para la compilación se incluyen junto al resto de los archivos fuentes del presente trabajo.

La compilación del programa se puede realizar de dos formas:

- *make generic*: compila la versión para *C* del programa. Utiliza *proximo.c*;
- *make mips*: compila la versión para *MIPS* del programa. Utiliza para ello *proximo.S*.

# 3. Utilización

Veamos ahora la forma en la que debe ser ejecutado el programa implementado en lenguaje C. El resultado de la compilación con "make" será un programa ejecutable, de nombre *autcel*, que podrá ser invocado con los siguientes parámetros:

- *-h*: Imprime ayuda para la utilización del programa;
- *-V*: Imprimer la versión actual del programa;
- *-o [Path]*: Especifica la ruta del archivo de salida sobre el cual se guarda la evolución generada por la aplicación.

Ejemplos:

```
$ ./autcel 30 80 inicial -o evolucion
```

Calcula la evolucion del automata Regla 30, en un mundo unidimensional de 80 celdas por 80 iteraciones. El archivo de salida se llama evolucion.pbm

# 4. Resultados Obtenidos

A continuación se muestran los resultados obtenidos para algunos casos de prueba:

Para el caso de la regla de 94 en 100 pasos vemos lo siguiente:

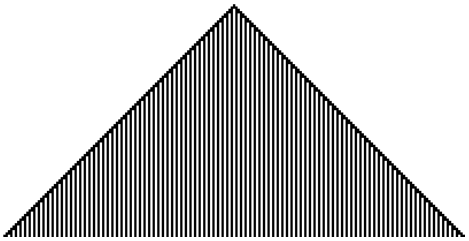
94 de 100x100.png 

**Figura 1:** *Imágen de la regla 94 aplicada en 100 pasos.*

Para el caso de la regla de 188 en 80 pasos vemos lo siguiente:

188 de 80x80.png 

**Figura 2:** *Imágen de la regla 188 aplicada en 80 pasos.*

Para el caso de la regla de 127 en 95 pasos vemos lo siguiente:

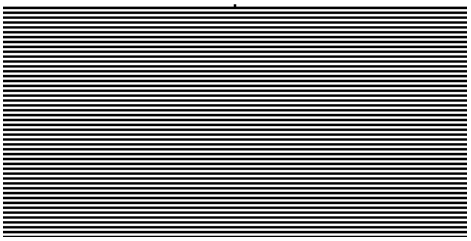
127 de 95x95.png 

**Figura 3:** *Imágen de la regla 127 aplicada en 95 pasos.*

# 5. Conclusiones

Las herramientas necesarias para realizar este trabajo resultaron dentro de todo simples de utilizar. Sin embargo, tuvimos y existen todavía problemas con la función próximo() implementada en assembly.

La integración entre el código en *C* y el *MIPS* no fue tan compleja y de hecho resultó interesante de realizar.

También tuvimos la necesidad de debuggear el programa hecho en MIPS, ya que al tener errores era necesario chequear el código paso a paso. Esto fue realizado con el *gdb* y fue de muchísima utilidad a la hora de encontrar errores.

# 6. Próximas Mejoras

Se espera para la próxima entrega poder tener implementada correctamente la función próximo en assembly.

## Referencias


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

[2] GCC, the GNU Compiler Collection, `http://gcc.gnu.org/`

[3] PBM format speciffication, `http://netpbm.sourceforge.net/doc/pbm.html`

[4] J. L. Hennessy and D. A. Patterson, "Computer Architecture. A Quantitative Approach," 4th Edition, Morgan Kaufmann Publishers, 2000.

# Apéndices

## A. Implementación completa en lenguaje C

### A.1. *main.c.* Implementación del main del programa

```c
#include <stdio.h>
#include <stdlib.h>
#include <plotter.h>
#include <string.h>

#define VERSION "1.0"

static void
do_usage();

void truncate_file_name(char* fileName);

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

    if(strcmp(argv[1],"-h")==0 || strcmp(argv[1],"--help")==0){
        do_usage();
        exit(0);
    }

    if (strcmp(argv[1], "-V") == 0 || strcmp(argv[1], "--version") == 0) {
        fprintf(stdout, "Version %s\n", VERSION);
        exit(0);
    }

    if(argc < 3 || argc > 6){
        fprintf(stderr, "Cantidad de parametros incorrectos.\n");
        exit(1);
    }

    char * rule = argv[1];
    int int_rule = atoi(rule);

    if(int_rule <= 0 || int_rule > 255){
        fprintf(stderr, "No se encontro una regla valida.\n");
        exit(1);
    }

    char * dim = argv[2];
    int int_dim = atoi(dim);

    if (int_dim <= 0 || int_dim > 1080) {
        fprintf(stderr, "No se encontro una cantidad de celdas validas.\n");
        exit(1);
    }

    char* input_file_name = argv[3];
    char* output_file_name = (char*) malloc(200);
    char buf[200];
    strcpy(output_file_name, input_file_name);
    if(argc==6 && strcmp(argv[4],"-o")==0){
        output_file_name = argv[5];
    }else{
        truncate_file_name(output_file_name);
    }
    snprintf(buf, sizeof buf, "%s%s", output_file_name, ".pbm");
    strcpy(output_file_name,buf);

    FILE *output;
    FILE *input;

    if (!(input = fopen(input_file_name, "r"))) {
        fprintf(stderr, "No se pudo abrir el archivo de entrada.\n");
        exit(1);
    }
```

```c

    if (!(output = fopen(output_file_name, "w"))) {
        fclose(input);
        fprintf(stderr, "No se pudo abrir el archivo de salida.\n");
        exit(1);
    }
    plotter_params_t params;
    params.output_file_pointer = output;
    params.input_file_pointer = input;
    params.width = int_dim;
    params.height = int_dim;
    params.rule = int_rule;
    plot(&params);
    return 0;
}

static void
do_usage()
{

    fprintf(stdout, "Uso:\n");
    fprintf(stdout, "autcel -h\n");
    fprintf(stdout, "autcel -V\n");
    fprintf(stdout, "autcel R N inputfile [-o outputprefix]\n");
    fprintf(stdout, "Opciones:\n");
    fprintf(stdout, "-h, --help\n");
    fprintf(stdout, "Imprime este mensaje.\n");
    fprintf(stdout, "-V, --version Da la version del programa\n");
    fprintf(stdout, "-o Prefijo de los archivos de salida.\n");
    fprintf(stdout, "Ejemplos:\n");
    fprintf(stdout, "autcel 30 80 inicial -o evolucion");
    fprintf(stdout, "Calcula la evolucion del automata Regla 30 \n");
    fprintf(stdout,
            "en un mundo unidimensional de 80 celdas, por 80 iteraciones.\n");
    fprintf(stdout, "El archivo de salida se llamara evoluci n.pbm\n");
    fprintf(stdout, "Si no se da un prefijo para los archivos de salida,\n");
    fprintf(stdout, "el prefijo sera  el  nombre del archivo de entrada.\n");
}

void truncate_file_name(char* fileName){
    int i = 0;
    while(fileName[i]!='\0' && i<100){
        if(fileName[i]=='.'){
            fileName[i]='\0';
            return;
        }
        i++;
    }
}
```

**Código 1:** *"main.c"*

## A.2. *plotter.h.* Declaración del algoritmo Bubblesort

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

typedef struct {
    size_t width;
    size_t height;
    unsigned char rule;

    FILE *output_file_pointer;
    FILE *input_file_pointer;
} plotter_params_t;

void
plot(plotter_params_t* params);

unsigned int
```

```
convertRule(unsigned char number, char result[8]);

char
decideNextChar(unsigned int previous, unsigned int current,unsigned int next, char result[8]);

unsigned char next_portable
(unsigned char *a, unsigned int i, unsigned int j, unsigned char regla, unsigned int N);
```

**Código 2:** *"plotter.h"*

## A.3. *plotter.c.* Definición del algoritmo Bubblesort

```
/*
 * plotter.c
 *
 *  Created on: 13 de set. de 2015
 *      Author: marcos
 */
#include <stdio.h>
#include <stdlib.h>
#include <plotter.h>
#include <proximo.h>

void plot(plotter_params_t* params) {
    fprintf(params->output_file_pointer, "P1\n");
    fprintf(params->output_file_pointer, "%u ", (unsigned) params->width);
    fprintf(params->output_file_pointer, "%u\n", (unsigned) params->height);

    size_t width = params->width;
    size_t height = params->height;
    int i, j = 0;

    char first_line[width * 2];
    unsigned char matrix[height][width];
    if (fgets(first_line, width * 2, params->input_file_pointer) == NULL) {
        fprintf(stderr, "Archivo de entrada incorrecto");
        fclose(params->output_file_pointer);
        fclose(params->input_file_pointer);
        exit(1);
    }

    i=0;
    fprintf(stdout, "Leyendo estado inicial...\n");
    for (j = 0; j < width * 2; j += 2) {
        fprintf(params->output_file_pointer, "%c", first_line[j]);
        matrix[0][i] = first_line[j];
        if (j < width * 2 - 2) {
            fprintf(params->output_file_pointer, " ");
        }
        i++;
    }
    fprintf(params->output_file_pointer, "\n");

    fprintf(stdout, "Grabando archivo de salida...\n");
    for (i = 1; i < height; i++) {
        for (j = 0; j < width; j++) {
            char to_print = '0';

            to_print = proximo(&matrix[0][0], i, j,params->rule, (unsigned int)width);
            fprintf(params->output_file_pointer, "%c",to_print);
            matrix[i][j] = to_print;
            if (j < width - 1) {
                fprintf(params->output_file_pointer, " ");
            }
        }
        if (i < height) {
            fprintf(params->output_file_pointer, "\n");
        }
    }
    if (fclose(params->output_file_pointer)) {
```

```
        fprintf(stderr, "No se pudo cerrar archivo.\n");
        exit(1);
    }
    if (fclose(params->input_file_pointer)) {
        fprintf(stderr, "No se pudo cerrar archivo.\n");
        exit(1);
    }
    fprintf(stdout, "Listo.\n");
}

unsigned int convertRule(unsigned char number, char result[8]) {

    unsigned int i = 0;
    for (i = 0; i < 8; i++) {
        result[i] = '0';
    }

    int rem;
    unsigned int size = 7;
    while (number != 0) {
        rem = number % 2;
        number /= 2;
        result[size] = rem + '0';
        size--;
    }
    result[8] = '\0';
    return 1;
}
```

**Código 3:** *"plotter.c"*

## A.4. *proximo.h*. Declaración del algoritmo Heapsort

```
/*
 * proximo.h
 *
 *  Created on: 23 de set. de 2015
 *      Author: marcos
 */

#ifndef PROXIMO_H_
#define PROXIMO_H_

#include <stdio.h>
#include <stdlib.h>

unsigned char proximo
(unsigned char *a, unsigned int i, unsigned int j, unsigned char regla, unsigned int N);


#endif /* PROXIMO_H_ */
```

**Código 4:** *"proximo.h"*

## A.5. *proximo.c*. Definición del algoritmo Heapsort

```
#include <proximo.h>

char decideNextChar(unsigned int previous, unsigned int current,
        unsigned int next, char rule[8]) {
    char result = '0';
    if (!previous && !current && !next) { //    000
        result = rule[7];
    }
```

```
    if (!previous && !current && next) { // 001
        result = rule[6];
    }
    if (!previous && current && !next) { // 010
        result = rule[5];
    }
    if (!previous && current && next) { //  011!previous && current && next
        result = rule[4];
    }
    if (previous && !current && !next) { // 100
        result = rule[3];
    }
    if (previous && !current && next) { //  101
        result = rule[2];
    }
    if (previous && current && !next) { //  110
        result = rule[1];
    }
    if (previous && current && next) { //       111
        result = rule[0];
    }
    return result;
}

unsigned char proximo(unsigned char *a, unsigned int i, unsigned int j,
        unsigned char regla, unsigned int N) {

    char rule[8];
    if(convertRule(regla,rule)==-1){
        fprintf(stderr, "Incorrect rules\n");
        exit(1);
    }

    unsigned int previousIndex = 0;
    unsigned int currentIndex = j;
    unsigned int nextIndex = 0;
    if (j == 0) {
        previousIndex = N - 1;
    } else {
        previousIndex = j - 1;
    }
    if (j == N) {
        nextIndex = 0;
    } else {
        nextIndex = j + 1;
    }
    return decideNextChar(a[(i - 1) * N + previousIndex] == '1',
            a[(i - 1)* N + currentIndex] == '1', a[ (i - 1)*N + nextIndex] == '1', rule);
}
```

**Código 5:** *"proximo.c"*

# B. Implementación completa en lenguaje Assembly MIPS32

## B.1. *proximo.S*. Definición del algoritmo Heapsort

```
#include <mips/regdef.h>
#include <sys/syscall.h>

# proximo.s :   Devuelve el estado del elemento en la proxima iteracion
#
# Parametros:
#               - a0 -> dir del array
#               - a1 -> int i (fila)
#               - a2 -> int j (columna)
#               - a3 -> regla
# Auxiliares:
#               - t0 -> N (tamanio de la matriz cuadrada)
#               - t1 -> dir inicial de la fila que necesito
#               - t2 -> direccion bloque izq
#               - t3 -> direccion bloque
#               - t4 -> direccion bloque der
#               - t5 -> aux
#               - t6 -> bloque izq
#               - t7 -> bloque
#               - t8 -> bloque der
#               - t9 -> regla

.text
.align 2
.globl proximo
.ent proximo


proximo:        subu sp, sp, 16 # Se crea el SRA 16 y LTA 0 y ABA 0 = 16 bytes
                sw ra, 8(sp)
                sw $fp, 4(sp)
                sw gp, 0(sp)
                move $fp, sp

                #Obtengo el valor de N
                lw t0, 32(sp) # t0 = N

                #Guardo la regla en t9
                addu t9, zero, a3 # t9 = regla

                #Obtengo la fila que necesito
                subu t1, t0, zero # t1 = N
                mul t1, a1, t1 # t1 = i * N
                mul t1, t1, 8 # t1 = i * N * 8
                addu t1, t1, a0 # t1 = a + [i * N]

                #Guardo el numero de bloque en cuestion
                addu t3, a2, zero # t3 = j

                #Si estoy al inicio del bloque, el hno izquierdo esta en el bloque anterior
                beqz a2, faltaHnoIzq

                #Si estoy al final del bloque, necesito al hno derecho que esta en el bloque siguiente
                subu t5, t0, 1 # t5 = N - 1
                beq a2, t5, faltaHnoDer

                #Si estoy en el medio del bloque, tengo los hnos que necesito
                b tengoLosHnos

faltaHnoIzq:    #En t4 guardo el ultimo nro de bloque
                subu t2, t0, 1 # t2 = N - 1
                addu t4, t3, 1 # t4 = 1
                b obtenerBloque # Voy a obtener el bloque

faltaHnoDer:    #En t4 guardo el primer nro de bloque
                subu t2, t3, 1 # t2 = N - 2
                addu t4, zero, zero # t4 = 0
                b obtenerBloque # Voy a obtener el bloque

tengoLosHnos:   #Guardo los hnos
```

```
                subu t2, t3, 1 # t2 = t3 - 1
                addu t4, t3, 1 # t4 = t3 + 1
                b obtenerBloque # Voy a obtener el bloque

obtenerBloque:  mul t2, t2, 8 # t2 = t2 * 8 -> Dir del bloque
                addu t5, t2, t1 # t5 = dir del bloque + inicio de la fila de matriz
                lb t6, 0(t5) # t6 = el bloque izquierdo
                subu t6, t6, 48 # Va a 0 o 1
                sll t6, t6, 2 # Corro a donde necesito al hno izq

                mul t3, t3, 8 # t3 = t3 * 8 -> Dir del bloque
                addu t5, t3, t1 # t5 = dir del bloque + inicio de la fila de matriz
                lb t7, 0(t5) # t7 = el bloque en cuestion
                subu t7, t7, 48 # Va a 0 o 1
                sll t7, t7, 1 # Corro a donde necesito al bit en cuestion

                mul t4, t4, 8 # t4 = t4 * 8 -> Dir del bloque
                addu t5, t4, t1 # t5 = dir del bloque + inicio de la fila de matriz
                lb t8, 0(t5) # t8 = el bloque derecho
                subu t8, t8, 48 # Va a 0 o 1

                #Uno a los 3 elementos en 1 para despues buscar la regla
                or t5, t6, t7
                or t5, t5, t8
                beq t5, zero, finRegla # Si es cero, no tengo que tocar regla, salto directo

aplicarRegla:   # Hay que aplicar la regla
                srl t9, t9, 1 # me quedo en el bit menos significativo el dato que necesito
                subu t5, t5, 1 # t5 = t5 - 1
                bne t5, zero, aplicarRegla
finRegla:       and t9, t9, 1 # pongo el resto en 0

devuelvoRes:    addu v0, t9, zero # v0 = estado de la proxima iteracion

salir:          lw ra, 8(sp)
                lw $fp, 4(sp)
                lw gp, 0(sp)
                addu sp, sp, 16
                jr ra
.end            proximo
```

**Código 6:** *"proximo.S"*