

# 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

<b>1. Introducción</b>	<b>2</b>
<b>2. Compilación</b>	<b>2</b>
<b>3. Utilización</b>	<b>2</b>
<b>4. Resultados Obtenidos</b>	<b>2</b>
<b>5. Conclusiones</b>	<b>3</b>
<b>6. Próximas Mejoras</b>	<b>3</b>
<b>Appendices</b>	<b>5</b>
<b>A. Implementación completa en lenguaje C</b>	<b>5</b>
A.1. <i>main.c</i> . Implementación del main del programa . . . . .	5
A.2. <i>plotter.h</i> . Declaración del algoritmo Bubblesort . . . . .	6
A.3. <i>plotter.c</i> . Definición del algoritmo Bubblesort . . . . .	7
A.4. <i>proximo.h</i> . Declaración del algoritmo Heapsort . . . . .	8
A.5. <i>proximo.c</i> . Definición del algoritmo Heapsort . . . . .	8
<b>B. Implementación completa en lenguaje Assembly MIPS32</b>	<b>10</b>
B.1. <i>proximo.S</i> . Definición del algoritmo Heapsort . . . . .	10

---

## 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.

## 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*: Imprimir 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
```

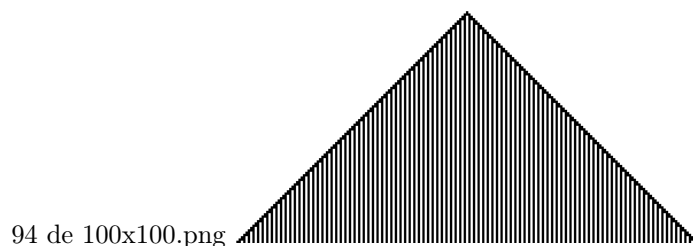
Calcula la evolucion del automata Regla 30, en un mundo unidimensional de 80 celdas por 80 iteraciones.

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:



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

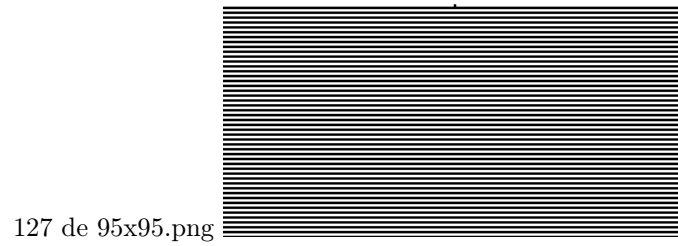
Para el caso de la regla de 188 en 80 pasos vemos lo siguiente:



**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:



**Figura 3:** *Imagen 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 specification, <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

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <plotter.h>
4 #include <string.h>
5
6 #define VERSION "1.0"
7
8 static void
9 do_usage();
10
11 void truncate_file_name(char* fileName);
12
13 int main(int argc, char * const argv[], char * const envp[]) {
14
15     if(strcmp(argv[1], "-h")==0 || strcmp(argv[1], "--help")==0){
16         do_usage();
17         exit(0);
18     }
19
20     if (strcmp(argv[1], "-V") == 0 || strcmp(argv[1], "--version") == 0) {
21         fprintf(stdout, "Version %s\n", VERSION);
22         exit(0);
23     }
24
25     if(argc < 3 || argc > 6){
26         fprintf(stderr, "Cantidad de parametros incorrectos.\n");
27         exit(1);
28     }
29
30     char * rule = argv[1];
31     int int_rule = atoi(rule);
32
33     if(int_rule <= 0 || int_rule > 255){
34         fprintf(stderr, "No se encontro una regla valida.\n");
35         exit(1);
36     }
37
38     char * dim = argv[2];
39     int int_dim = atoi(dim);
40
41     if (int_dim <= 0 || int_dim > 1080) {
42         fprintf(stderr, "No se encontro una cantidad de celdas validas.\n");
43         exit(1);
44     }
45
46     char* input_file_name = argv[3];
47     char* output_file_name = (char*) malloc(200);
48     char buf[200];
49     strcpy(output_file_name, input_file_name);
50     if(argc==6 && strcmp(argv[4], "-o")==0){
51         output_file_name = argv[5];
52     }else{
53         truncate_file_name(output_file_name);
54     }
55     snprintf(buf, sizeof buf, "%s%s", output_file_name, ".pbm");
56     strcpy(output_file_name, buf);
57
58     FILE *output;
59     FILE *input;
60
61     if (!(input = fopen(input_file_name, "r"))) {
62         fprintf(stderr, "No se pudo abrir el archivo de entrada.\n");
63         exit(1);
64     }
```

```

65
66     if (!(output = fopen(output_file_name, "w"))) {
67         fclose(input);
68         fprintf(stderr, "No se pudo abrir el archivo de salida.\n");
69         exit(1);
70     }
71     plotter_params_t params;
72     params.output_file_pointer = output;
73     params.input_file_pointer = input;
74     params.width = int_dim;
75     params.height = int_dim;
76     params.rule = int_rule;
77     plot(&params);
78     return 0;
79 }
80
81 static void
82 do_usage()
83 {
84
85     fprintf(stdout, "Uso:\n");
86     fprintf(stdout, "autcel -h\n");
87     fprintf(stdout, "autcel -V\n");
88     fprintf(stdout, "autcel R N inputfile [-o outputprefix]\n");
89     fprintf(stdout, "Opciones:\n");
90     fprintf(stdout, "-h, --help\n");
91     fprintf(stdout, "Imprime este mensaje.\n");
92     fprintf(stdout, "-V, --version Da la version del programa\n");
93     fprintf(stdout, "-o Prefijo de los archivos de salida.\n");
94     fprintf(stdout, "Ejemplos:\n");
95     fprintf(stdout, "autcel 30 80 inicial -o evolucion");
96     fprintf(stdout, "Calcula la evolucion del automata Regla 30 \n");
97     fprintf(stdout,
98         "en un mundo unidimensional de 80 celdas, por 80 iteraciones.\n");
99     fprintf(stdout, "El archivo de salida se llamara evoluci n.pbm\n");
100    fprintf(stdout, "Si no se da un prefijo para los archivos de salida,\n");
101    fprintf(stdout, "el prefijo sera el nombre del archivo de entrada.\n");
102 }
103
104 void truncate_file_name(char* fileName){
105     int i = 0;
106     while(fileName[i]!='\0' && i<100){
107         if(fileName[i]=='.'){
108             fileName[i]='\0';
109             return;
110         }
111         i++;
112     }
113 }

```

Código 1: “main.c”

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

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct {
5     size_t width;
6     size_t height;
7     unsigned char rule;
8
9     FILE *output_file_pointer;
10    FILE *input_file_pointer;
11 } plotter_params_t;
12
13 void
14 plot(plotter_params_t* params);
15
16 unsigned int

```

```

17 convertRule(unsigned char number, char result[8]);
18
19 char
20 decideNextChar(unsigned int previous, unsigned int current, unsigned int next, char result[8]);
21
22 unsigned char next_portable
23 (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

```

1  /*
2  * plotter.c
3  *
4  * Created on: 13 de set. de 2015
5  * Author: marcos
6  */
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <plotter.h>
10 #include <proximo.h>
11
12 void plot(plotter_params_t* params) {
13     fprintf(params->output_file_pointer, "P1\n");
14     fprintf(params->output_file_pointer, "%u ", (unsigned) params->width);
15     fprintf(params->output_file_pointer, "%u\n", (unsigned) params->height);
16
17     size_t width = params->width;
18     size_t height = params->height;
19     int i, j = 0;
20
21     char first_line[width * 2];
22     unsigned char matrix[height][width];
23     if (fgets(first_line, width * 2, params->input_file_pointer) == NULL) {
24         fprintf(stderr, "Archivo de entrada incorrecto");
25         fclose(params->output_file_pointer);
26         fclose(params->input_file_pointer);
27         exit(1);
28     }
29
30     i=0;
31     fprintf(stdout, "Leyendo estado inicial...\n");
32     for (j = 0; j < width * 2; j += 2) {
33         fprintf(params->output_file_pointer, "%c", first_line[j]);
34         matrix[0][i] = first_line[j];
35         if (j < width * 2 - 2) {
36             fprintf(params->output_file_pointer, " ");
37         }
38         i++;
39     }
40     fprintf(params->output_file_pointer, "\n");
41
42     fprintf(stdout, "Grabando archivo de salida...\n");
43     for (i = 1; i < height; i++) {
44         for (j = 0; j < width; j++) {
45             char to_print = '0';
46
47             to_print = proximo(&matrix[0][0], i, j, params->rule, (unsigned int)width);
48             fprintf(params->output_file_pointer, "%c", to_print);
49             matrix[i][j] = to_print;
50             if (j < width - 1) {
51                 fprintf(params->output_file_pointer, " ");
52             }
53         }
54         if (i < height) {
55             fprintf(params->output_file_pointer, "\n");
56         }
57     }
58     if (fclose(params->output_file_pointer)) {

```

```

59     fprintf(stderr, "No se pudo cerrar archivo.\n");
60     exit(1);
61 }
62 if (fclose(params->input_file_pointer)) {
63     fprintf(stderr, "No se pudo cerrar archivo.\n");
64     exit(1);
65 }
66 fprintf(stdout, "Listo.\n");
67 }
68
69 unsigned int convertRule(unsigned char number, char result[8]) {
70
71     unsigned int i = 0;
72     for (i = 0; i < 8; i++) {
73         result[i] = '0';
74     }
75
76     int rem;
77     unsigned int size = 7;
78     while (number != 0) {
79         rem = number % 2;
80         number /= 2;
81         result[size] = rem + '0';
82         size--;
83     }
84     result[8] = '\0';
85     return 1;
86 }

```

Código 3: “*plotter.c*”

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

```

1  /*
2  * proximo.h
3  *
4  * Created on: 23 de set. de 2015
5  * Author: marcos
6  */
7
8  #ifndef PROXIMO_H_
9  #define PROXIMO_H_
10
11 #include <stdio.h>
12 #include <stdlib.h>
13
14 unsigned char proximo
15 (unsigned char *a, unsigned int i, unsigned int j, unsigned char regla, unsigned int N);
16
17
18 #endif /* PROXIMO_H_ */

```

Código 4: “*proximo.h*”

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

```

1  #include <proximo.h>
2
3  char decideNextChar(unsigned int previous, unsigned int current,
4                     unsigned int next, char rule[8]) {
5      char result = '0';
6      if (!previous && !current && !next) { //    000
7          result = rule[7];
8      }

```



```

9     if (!previous && !current && next) { // 001
10         result = rule[6];
11     }
12     if (!previous && current && !next) { // 010
13         result = rule[5];
14     }
15     if (!previous && current && next) { // 011!previous && current && next
16         result = rule[4];
17     }
18     if (previous && !current && !next) { // 100
19         result = rule[3];
20     }
21     if (previous && !current && next) { // 101
22         result = rule[2];
23     }
24     if (previous && current && !next) { // 110
25         result = rule[1];
26     }
27     if (previous && current && next) { // 111
28         result = rule[0];
29     }
30     return result;
31 }
32
33 unsigned char proximo(unsigned char *a, unsigned int i, unsigned int j,
34     unsigned char regla, unsigned int N) {
35
36     char rule[8];
37     if(convertRule(regla,rule)==-1){
38         fprintf(stderr, "Incorrect rules\n");
39         exit(1);
40     }
41
42     unsigned int previousIndex = 0;
43     unsigned int currentIndex = j;
44     unsigned int nextIndex = 0;
45     if (j == 0) {
46         previousIndex = N - 1;
47     } else {
48         previousIndex = j - 1;
49     }
50     if (j == N) {
51         nextIndex = 0;
52     } else {
53         nextIndex = j + 1;
54     }
55     return decideNextChar(a[(i - 1) * N + previousIndex] == '1',
56         a[(i - 1) * N + currentIndex] == '1', a[(i - 1) * N + nextIndex] == '1', rule);
57 }

```

**Código 5:** “*proximo.c*”

## B. Implementación completa en lenguaje Assembly MIPS32

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

```
1 #include <mips/regdef.h>
2 #include <sys/syscall.h>
3
4 # proximo.s :   Devuelve el estado del elemento en la proxima iteracion
5 #
6 # Parametros:
7 #             - a0 -> dir del array
8 #             - a1 -> int i (fila)
9 #             - a2 -> int j (columna)
10 #            - a3 -> regla
11 # Auxiliares:
12 #            - t0 -> N (tamano de la matriz cuadrada)
13 #            - t1 -> dir inicial de la fila que necesito
14 #            - t2 -> direccion bloque izq
15 #            - t3 -> direccion bloque
16 #            - t4 -> direccion bloque der
17 #            - t5 -> aux
18 #            - t6 -> bloque izq
19 #            - t7 -> bloque
20 #            - t8 -> bloque der
21 #            - t9 -> regla
22
23 .text
24 .align 2
25 .globl proximo
26 .ent proximo
27
28
29 proximo:      subu sp, sp, 16 # Se crea el SRA 16 y LTA 0 y ABA 0 = 16 bytes
30              sw ra, 8(sp)
31              sw $fp, 4(sp)
32              sw gp, 0(sp)
33              move $fp, sp
34
35              #Obtengo el valor de N
36              lw t0, 32(sp) # t0 = N
37
38              #Guardo la regla en t9
39              addu t9, zero, a3 # t9 = regla
40
41              #Obtengo la fila que necesito
42              subu t1, t0, zero # t1 = N
43              mul t1, a1, t1 # t1 = i * N
44              mul t1, t1, 8 # t1 = i * N * 8
45              addu t1, t1, a0 # t1 = a + [i * N]
46
47              #Guardo el numero de bloque en cuestion
48              addu t3, a2, zero # t3 = j
49
50              #Si estoy al inicio del bloque, el hno izquierdo esta en el bloque anterior
51              beqz a2, faltaHnoIzq
52
53              #Si estoy al final del bloque, necesito al hno derecho que esta en el bloque siguiente
54              subu t5, t0, 1 # t5 = N - 1
55              beq a2, t5, faltaHnoDer
56
57              #Si estoy en el medio del bloque, tengo los hnos que necesito
58              b tengoLosHnos
59
60 faltaHnoIzq:  #En t4 guardo el ultimo nro de bloque
61              subu t2, t0, 1 # t2 = N - 1
62              addu t4, t3, 1 # t4 = 1
63              b obtenerBloque # Voy a obtener el bloque
64
65 faltaHnoDer:  #En t4 guardo el primer nro de bloque
66              subu t2, t3, 1 # t2 = N - 2
67              addu t4, zero, zero # t4 = 0
68              b obtenerBloque # Voy a obtener el bloque
69
70 tengoLosHnos: #Guardo los hnos
```

```

71         subu t2, t3, 1 # t2 = t3 - 1
72         addu t4, t3, 1 # t4 = t3 + 1
73         b obtenerBloque # Voy a obtener el bloque
74
75 obtenerBloque: mul t2, t2, 8 # t2 = t2 * 8 -> Dir del bloque
76               addu t5, t2, t1 # t5 = dir del bloque + inicio de la fila de matriz
77               lb t6, 0(t5) # t6 = el bloque izquierdo
78               subu t6, t6, 48 # Va a 0 o 1
79               sll t6, t6, 2 # Corro a donde necesito al hno izq
80
81               mul t3, t3, 8 # t3 = t3 * 8 -> Dir del bloque
82               addu t5, t3, t1 # t5 = dir del bloque + inicio de la fila de matriz
83               lb t7, 0(t5) # t7 = el bloque en cuestion
84               subu t7, t7, 48 # Va a 0 o 1
85               sll t7, t7, 1 # Corro a donde necesito al bit en cuestion
86
87               mul t4, t4, 8 # t4 = t4 * 8 -> Dir del bloque
88               addu t5, t4, t1 # t5 = dir del bloque + inicio de la fila de matriz
89               lb t8, 0(t5) # t8 = el bloque derecho
90               subu t8, t8, 48 # Va a 0 o 1
91
92               #Uno a los 3 elementos en 1 para despues buscar la regla
93               or t5, t6, t7
94               or t5, t5, t8
95               beq t5, zero, finRegla # Si es cero, no tengo que tocar regla, salto directo
96
97 aplicarRegla: # Hay que aplicar la regla
98               srl t9, t9, 1 # me quedo en el bit menos significativo el dato que necesito
99               subu t5, t5, 1 # t5 = t5 - 1
100              bne t5, zero, aplicarRegla
101 finRegla:      and t9, t9, 1 # pongo el resto en 0
102
103 devuelvoRes:  addu v0, t9, zero # v0 = estado de la proxima iteracion
104
105 salir:        lw ra, 8(sp)
106               lw $fp, 4(sp)
107               lw gp, 0(sp)
108               addu sp, sp, 16
109               jr ra
110 .end          proximo

```

**Código 6:** “*proximo.S*”