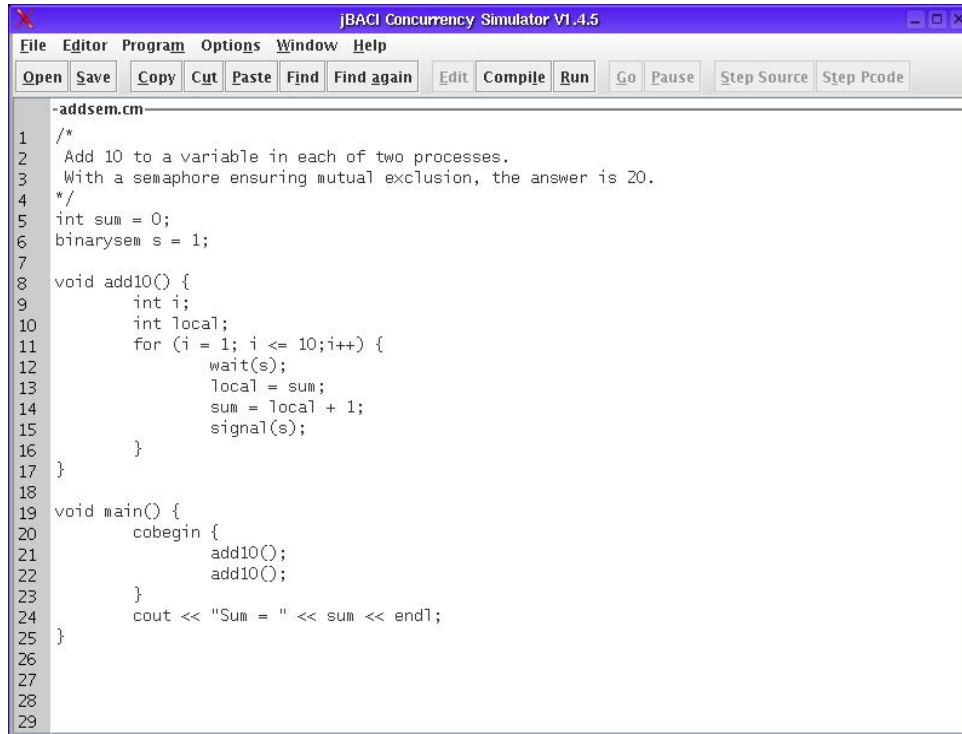




# JBACI

COMUNICACIÓN Y  
SINCRONIZACIÓN DE  
PROCESOS.

# JBACI



The screenshot shows the JBACI Concurrency Simulator V1.4.5 window. The menu bar includes File, Editor, Program, Options, Window, and Help. The toolbar contains buttons for Open, Save, Copy, Cut, Paste, Find, Find again, Edit, Compile, Run, Go, Pause, Step Source, and Step Pcode. The main editor area displays a C program named -addsem.cm. The program includes a comment explaining the task: adding 10 to a variable in two processes with mutual exclusion using a semaphore. The code defines a semaphore, a function add10(), and a main function that spawns two processes to call add10().

```
1  /*
2  Add 10 to a variable in each of two processes.
3  With a semaphore ensuring mutual exclusion, the answer is 20.
4  */
5  int sum = 0;
6  binarysem s = 1;
7
8  void add10() {
9      int i;
10     int local;
11     for (i = 1; i <= 10; i++) {
12         wait(s);
13         local = sum;
14         sum = local + 1;
15         signal(s);
16     }
17 }
18
19 void main() {
20     cobegin {
21         add10();
22         add10();
23     }
24     cout << "Sum = " << sum << endl;
25 }
26
27
28
29
```

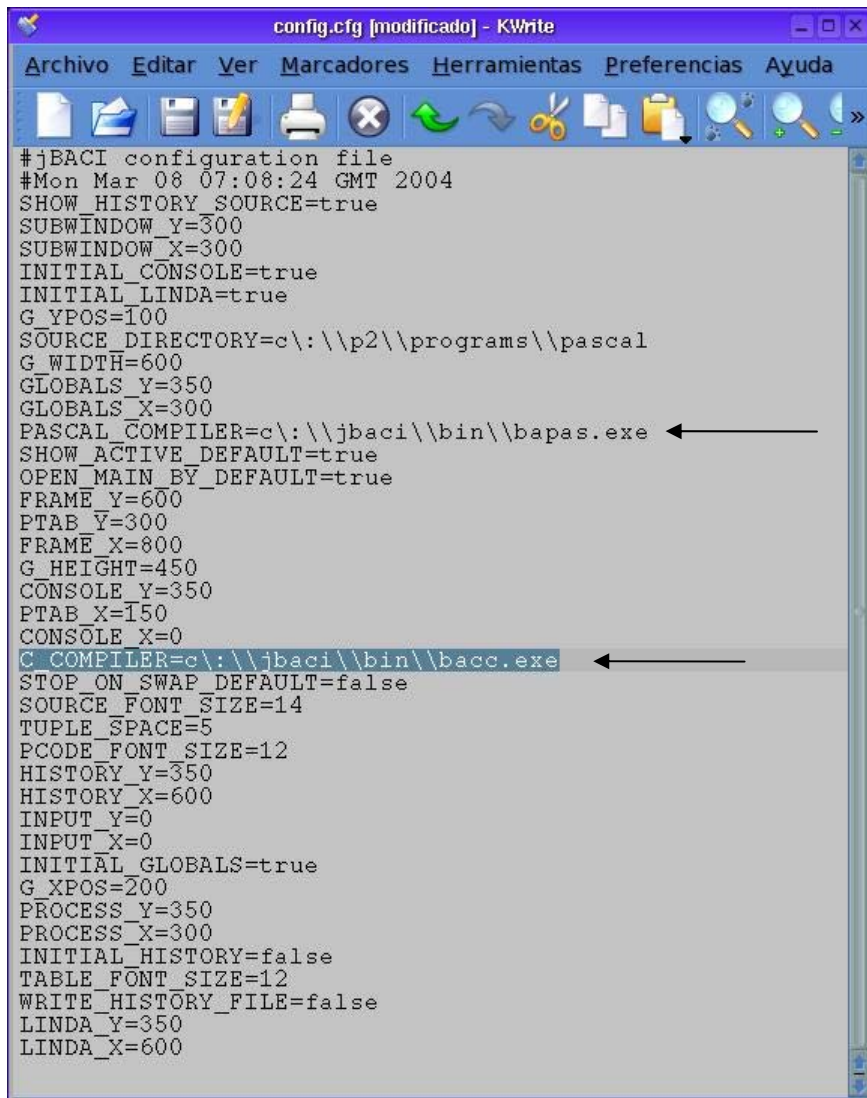
En esta imagen se ve a JBACI en ejecución con un sencillo programa ejemplo de semáforos.

- jBACI es un entorno de desarrollo integrado para el aprendizaje de programación concurrente mediante la simulación de concurrencia.
- Está construido sobre el compilador **BACI** y sobre el interprete **BACI Debugger**.
- Los compiladores BACI (tanto para Pascal como para C) traducen programas concurrentes en un lenguaje intermedio llamado **Pcode**.
- Los compiladores soportan primitivas de sincronización como **semáforos** y **monitores**.

# JBACI

- JBACI es un programa escrito en **Java** por lo que requiere de el **runtime** de Java para poder ejecutarse.
- Por estar escrito en Java es **multiplataforma**, lo que permite que lo podamos usar en distintos sistemas operativos, por ejemplo en Linux o en Windows.
- En la página web del proyecto disponen de los ejecutables de JBACI así como de las fuentes del mismo.
- Existe también documentación sobre el compilador BACI, bastante extensa aunque está en Inglés.
- Para el correcto funcionamiento del programa, se necesita ajustar el fichero de configuración que es creado la primera vez que se ejecuta el programa si no existía antes.
- El fichero se llama config.cfg y los cambios a hacer son indicarle las rutas de los programas compiladores (bacc y bapas) y opcionalmente la ruta de los ficheros fuentes.
- Esto podemos hacerlo con cualquier editor de textos que tengamos a mano pues el fichero contiene texto plano.
- Conviene recordar que las rutas en Linux y en Windows no se escriben de la misma manera.

# JBACI

A screenshot of a KWrite text editor window titled 'config.cfg [modificado] - KWrite'. The window has a menu bar with 'Archivo', 'Editar', 'Ver', 'Marcadores', 'Herramientas', 'Preferencias', and 'Ayuda'. Below the menu is a toolbar with various icons. The main text area contains a configuration file for JBACI. The file starts with a comment '#jBACI configuration file' and a timestamp '#Mon Mar 08 07:08:24 GMT 2004'. It lists various settings such as 'SHOW\_HISTORY\_SOURCE=true', 'SUBWINDOW\_Y=300', 'INITIAL\_CONSOLE=true', 'INITIAL\_LINDA=true', 'G\_YPOS=100', 'SOURCE\_DIRECTORY=c:\\p2\\programs\\pascal', 'G\_WIDTH=600', 'GLOBALS\_Y=350', 'GLOBALS\_X=300', 'PASCAL\_COMPILER=c:\\jbaci\\bin\\bapas.exe', 'SHOW\_ACTIVE\_DEFAULT=true', 'OPEN\_MAIN\_BY\_DEFAULT=true', 'FRAME\_Y=600', 'PTAB\_Y=300', 'FRAME\_X=800', 'G\_HEIGHT=450', 'CONSOLE\_Y=350', 'PTAB\_X=150', 'CONSOLE\_X=0', 'C\_COMPILER=c:\\jbaci\\bin\\bacc.exe', 'STOP\_ON\_SWAP\_DEFAULT=false', 'SOURCE\_FONT\_SIZE=14', 'TUPLE\_SPACE=5', 'PCODE\_FONT\_SIZE=12', 'HISTORY\_Y=350', 'HISTORY\_X=600', 'INPUT\_Y=0', 'INPUT\_X=0', 'INITIAL\_GLOBALS=true', 'G\_XPOS=200', 'PROCESS\_Y=350', 'PROCESS\_X=300', 'INITIAL\_HISTORY=false', 'TABLE\_FONT\_SIZE=12', 'WRITE\_HISTORY\_FILE=false', 'LINDA\_Y=350', and 'LINDA\_X=600'. Two arrows point to the 'PASCAL\_COMPILER' and 'C\_COMPILER' lines.

```
#jBACI configuration file
#Mon Mar 08 07:08:24 GMT 2004
SHOW_HISTORY_SOURCE=true
SUBWINDOW_Y=300
SUBWINDOW_X=300
INITIAL_CONSOLE=true
INITIAL_LINDA=true
G_YPOS=100
SOURCE_DIRECTORY=c:\\p2\\programs\\pascal
G_WIDTH=600
GLOBALS_Y=350
GLOBALS_X=300
PASCAL_COMPILER=c:\\jbaci\\bin\\bapas.exe
SHOW_ACTIVE_DEFAULT=true
OPEN_MAIN_BY_DEFAULT=true
FRAME_Y=600
PTAB_Y=300
FRAME_X=800
G_HEIGHT=450
CONSOLE_Y=350
PTAB_X=150
CONSOLE_X=0
C_COMPILER=c:\\jbaci\\bin\\bacc.exe
STOP_ON_SWAP_DEFAULT=false
SOURCE_FONT_SIZE=14
TUPLE_SPACE=5
PCODE_FONT_SIZE=12
HISTORY_Y=350
HISTORY_X=600
INPUT_Y=0
INPUT_X=0
INITIAL_GLOBALS=true
G_XPOS=200
PROCESS_Y=350
PROCESS_X=300
INITIAL_HISTORY=false
TABLE_FONT_SIZE=12
WRITE_HISTORY_FILE=false
LINDA_Y=350
LINDA_X=600
```

- Una vez configurado el programa el siguiente paso es ejecutarlo.
- Para ello, si estamos en Linux, abriremos una consola y teclearemos el siguiente comando desde el directorio donde guardamos JBACI:
- `java -jar jbací.jar`
- Página web del proyecto JBACI: Página web para descargar Java runtime:  
<http://www.java.com/es/download/>
- Howto para Java en Linux:  
<http://www.educ.umu.se/~bjorn/linux/howto/Java-HOWTO.html>

# JBACI

File	
New	Ctrl-N
Open	Ctrl-O
Save	Ctrl-S
Save as	Ctrl-A
Exit	Ctrl-Q

Dentro de este menú encontramos los comandos básicos para administrar los archivos con los que estamos trabajando. Desde aquí podemos guardar y cargar diferentes archivos.

---

Editor	
Copy	Ctrl-C
Cut	Ctrl-X
Paste	Ctrl-V
Find	Ctrl-F
Find again	Ctrl-I

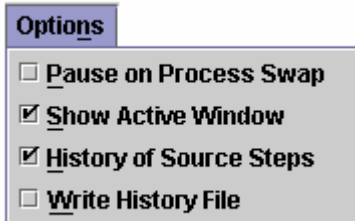
En el menú Editor contamos con diversas operaciones que nos facilitan la edición de nuestro documento, permitiéndonos duplicar partes del código fuente así como encontrar expresiones específicas dentro de nuestro programa.

---

Program	
Edit	Ctrl-E
Compile	Ctrl-L
Run	Ctrl-R
Go	Ctrl-G
Pause	Escape
Step Source	Introduzca
Step Pcode	Espacio

En este menú encontramos los comandos relacionados con el funcionamiento del programa. Desde aquí podemos realizar la compilación y posterior ejecución del programa.

# JBACI



Dentro de este apartado encontramos una serie de opciones que afectan a la ejecución del programa que podremos ajustar según nuestra preferencia. Por ejemplo la posibilidad de mostrar o no el historial.

---



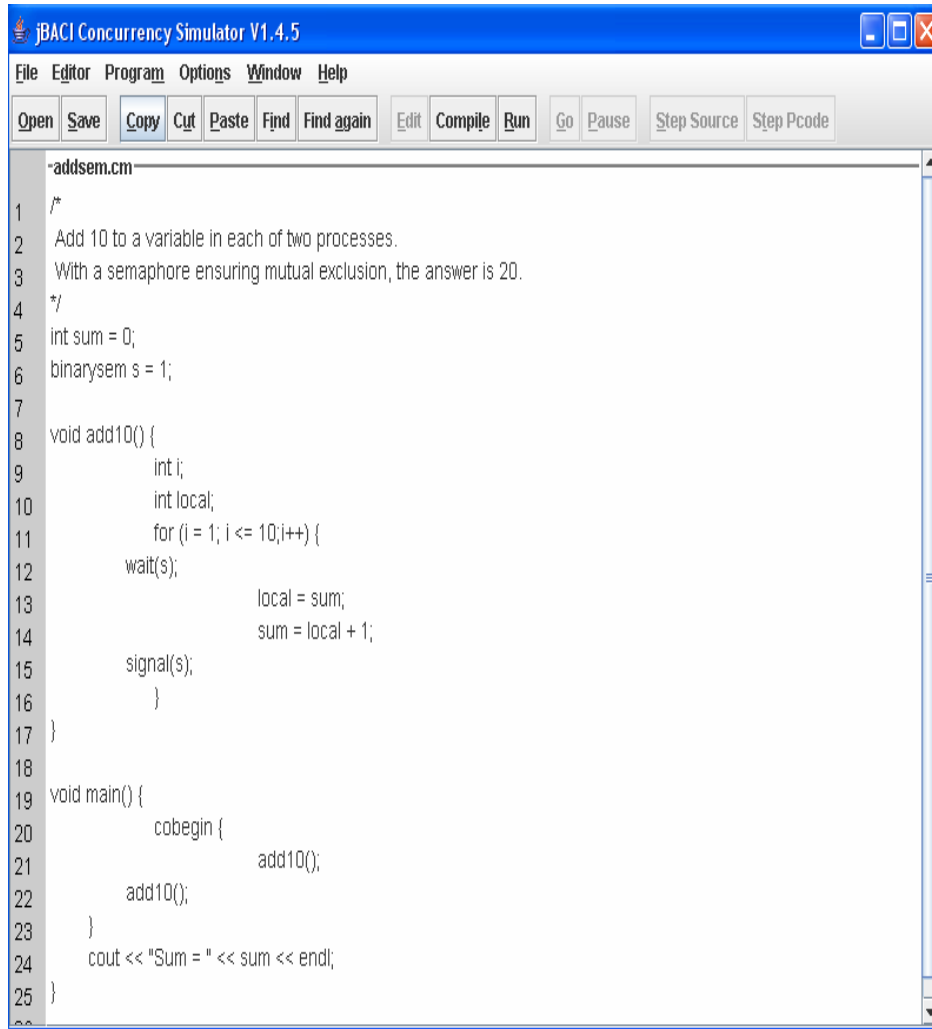
Dentro de este menú encontramos los comandos de selección de ventanas que aparecerán durante la ejecución, como veremos mas adelante.

---



En el menú Help encontraremos cierta información del programa relacionada con la versión que estamos utilizando o su carácter gratuito.

# JBACI

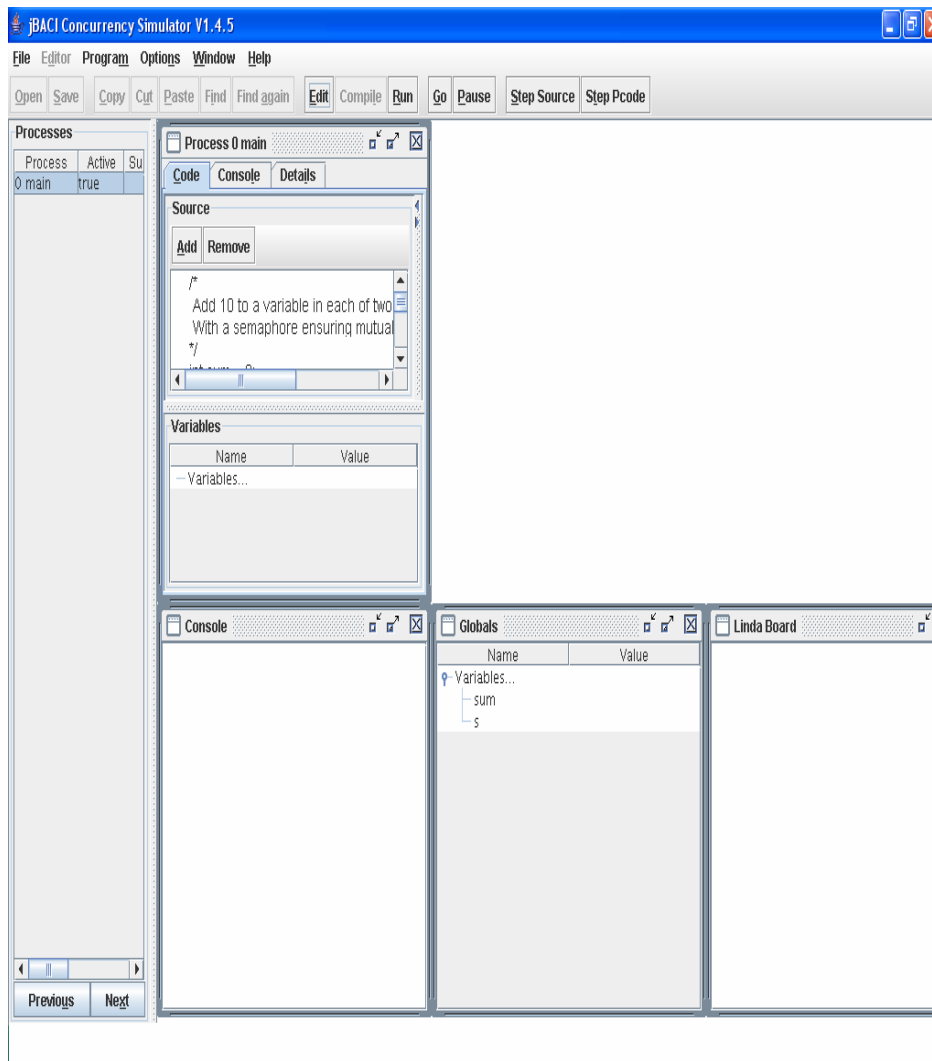


The screenshot shows the JBACI Concurrency Simulator V1.4.5 window. The title bar reads "JBACI Concurrency Simulator V1.4.5". The menu bar includes "File", "Editor", "Program", "Options", "Window", and "Help". The toolbar contains buttons for "Open", "Save", "Copy", "Cut", "Paste", "Find", "Find again", "Edit", "Compile", "Run", "Go", "Pause", "Step Source", and "Step Pcode". The main text area displays the source code for a file named "addsem.cm".

```
-addsem.cm-
1  /*
2   Add 10 to a variable in each of two processes.
3   With a semaphore ensuring mutual exclusion, the answer is 20.
4  */
5  int sum = 0;
6  binarysem s = 1;
7
8  void add10() {
9      int i;
10     int local;
11     for (i = 1; i <= 10; i++) {
12         wait(s);
13         local = sum;
14         sum = local + 1;
15         signal(s);
16     }
17 }
18
19 void main() {
20     cobegin {
21         add10();
22     }
23     cout << "Sum = " << sum << endl;
24 }
25 }
```

- El siguiente paso seria ver el funcionamiento del programa.
- Para ello jbaci nos proporciona una serie una serie de ejemplos que podemos utilizar. En este caso el addsem.cm.
- A continuación lo que haremos es compilar el programa. “Compile”
- .
- Si la compilación a sido satisfactoria presionaremos la opción “Run”.
- Seguidamente el programa nos abre una nueva ventana.

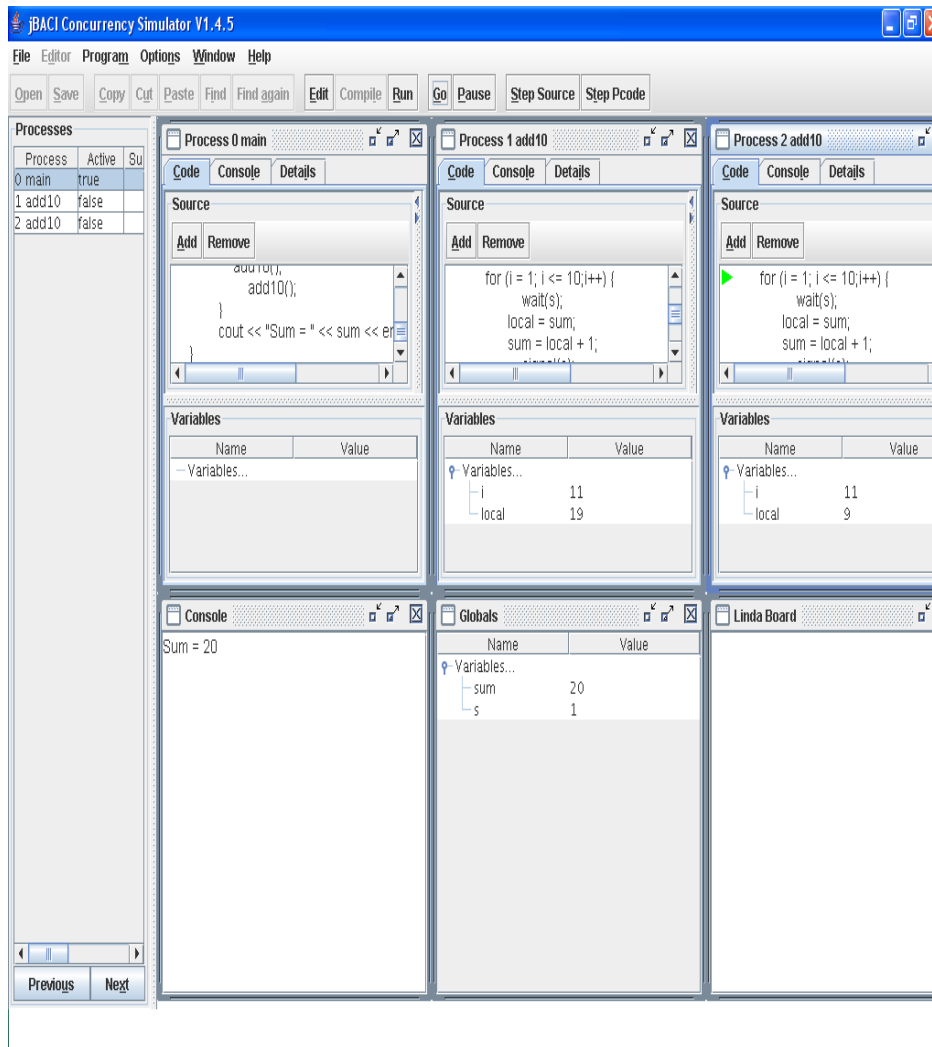
# JBACI



- Podemos destacar 5 zonas diferenciadas:
  - Processes: aquí el programa nos muestra información sobre los procesos que esta o han sido ejecutados.
  - Process: nos aparecerán tantas ventadas como procesos en ejecución tengamos. En estas ventadas nos aparecerá el estado del proceso y el valor de las variables locales. En nuestro caso (ejemplo addsem) nos aparecerán 3 ventadas una para el main y dos para add10.
  - Otra parte esencial es la consola: JBACI nos proporciona una consola que nos mostrara el resultado del proceso.
  - Globals: aquí nos ira apareciendo el valor de las variables globales.



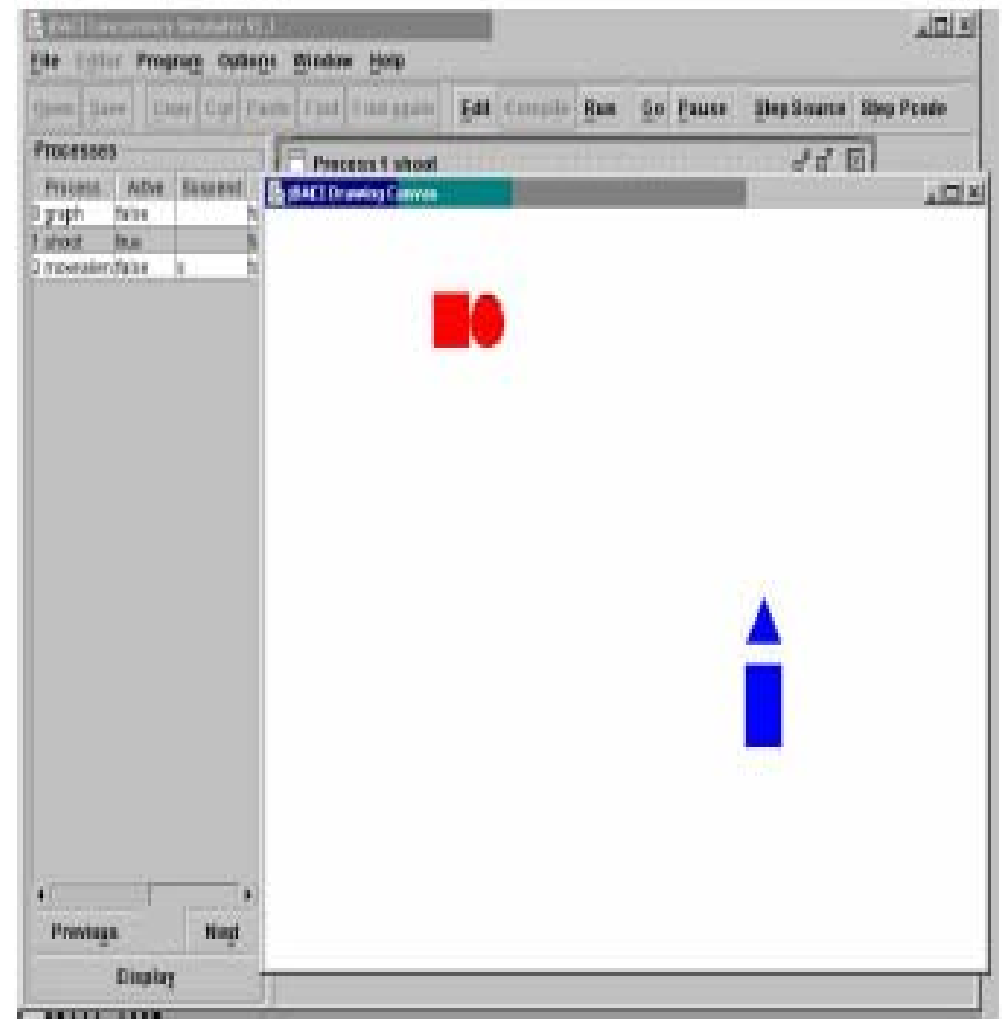
# JBACI



- En la imagen podemos ver la simulación finalizada.
- JBACI nos ofrece 2 formas para ejecutar la simulación:
  - Presionando la opción GO mediante la cual se simula el programa por completo, pudiéndolo parar en cualquier momento con la opción pause.
  - Y ejecutarlo paso a paso mediante las opciones step source y step pcode. Con estas opciones el programa se ejecutará paso a paso, dándonos la posibilidad de ver el valor de sus variables en todo momento.
- Si tenemos algún problema con la compilación podemos volver a la pantalla principal presionando edit.

# JBACI

- El API de Java es usado en jBACI para crear rutinas graficas utilizadas en programas concurrentes.
- Esta utilidad es muy útil para hacer comprobaciones de la correcta sincronización en nuestros programas.
- El siguiente ejemplo muestra como jBACI también posee comandos gráficos que podemos incluir en nuestro programa.



# JBACI

```
program Sirpinski;
{ Demonstration of jBACI graphics commands - Sirpinski space-filling. }
{ Written by Shmuel Schwartz. }

#include "qdefs.pm"
const w=450; h=450;
var rec_num: integer;
s:semaphore:=1;

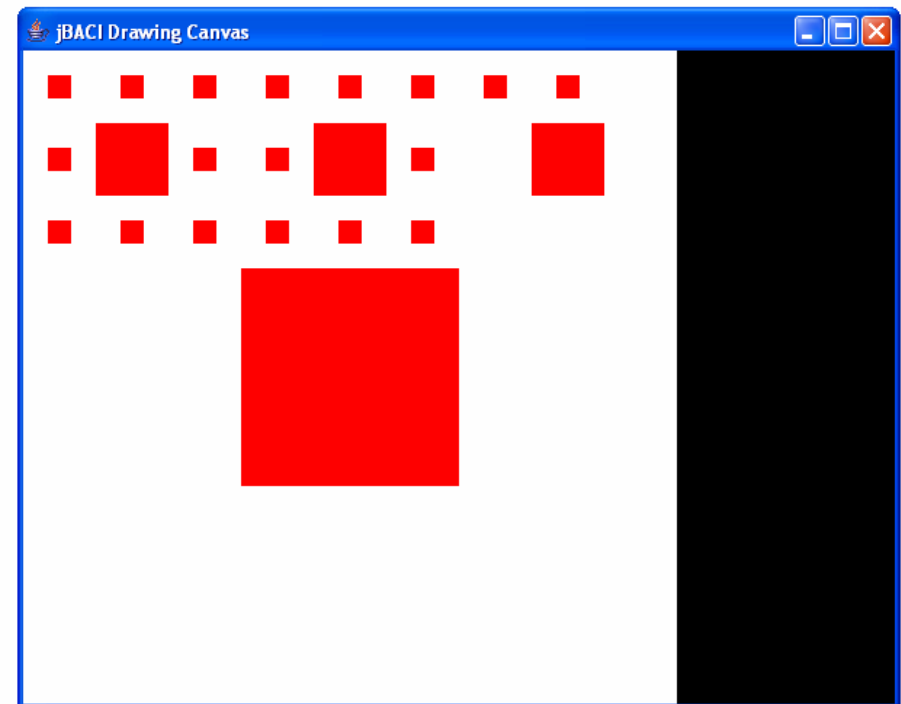
procedure subrec(x,y,l,i:integer);
begin
  if i>0 then
  begin
    create(rec_num,RECTANGLE,1,x-(l div 2),y-(l div 2),l,1);
    rec_num:=rec_num+1;
    subrec(x - l,y - l,l div 3,i-1);
    subrec(x,y - l,l div 3,i-1);
    subrec(x + l,y - l,l div 3,i-1);
    subrec(x - l,y,l div 3,i-1);
    subrec(x + l,y,l div 3,i-1);
    subrec(x - l,y + l,l div 3,i-1);
    subrec(x,y + l,l div 3,i-1);
    subrec(x + l,y + l,l div 3,i-1);
  end;
  writeln(i);
end;

begin
  rec_num:=1;
  create(0,RECTANGLE,white,0,0,600,600);
  create(1000,RECTANGLE,black,450,0,150,450);
  subrec(w div 2,h div 2,h div 3,3);
  writeln('ended ');
end.
```

- Dentro de la carpeta de ejemplo cargamos y compilamos el archivo con el nombre sir.pm
- Este programa dibujará una serie de figuras, en este caso rectangulos, en unas determinadas posiciones y tamaños.
- Si a continuación ejecutamos “GO” el programa abrirá una ventana donde comenzará a dibujar las figuras.

# JBACI

- Las posiciones y tamaños están expresadas en pixeles, la esquina izquierda superior es el punto (0,0) y la inferior derecha es el punto (600,450), aunque puede ser modificado en 'Config.java'.
- Todos los parámetros utilizados son de tipo entero y toda la codificación de las figuras y colores esta declarada como constantes en los ficheros 'gdefs.pm' y 'gdefs.cm'
- Cada figura se crea junto un identificador o 'handle' que servira para poder referirnos a ella en siguientes ejecuciones.



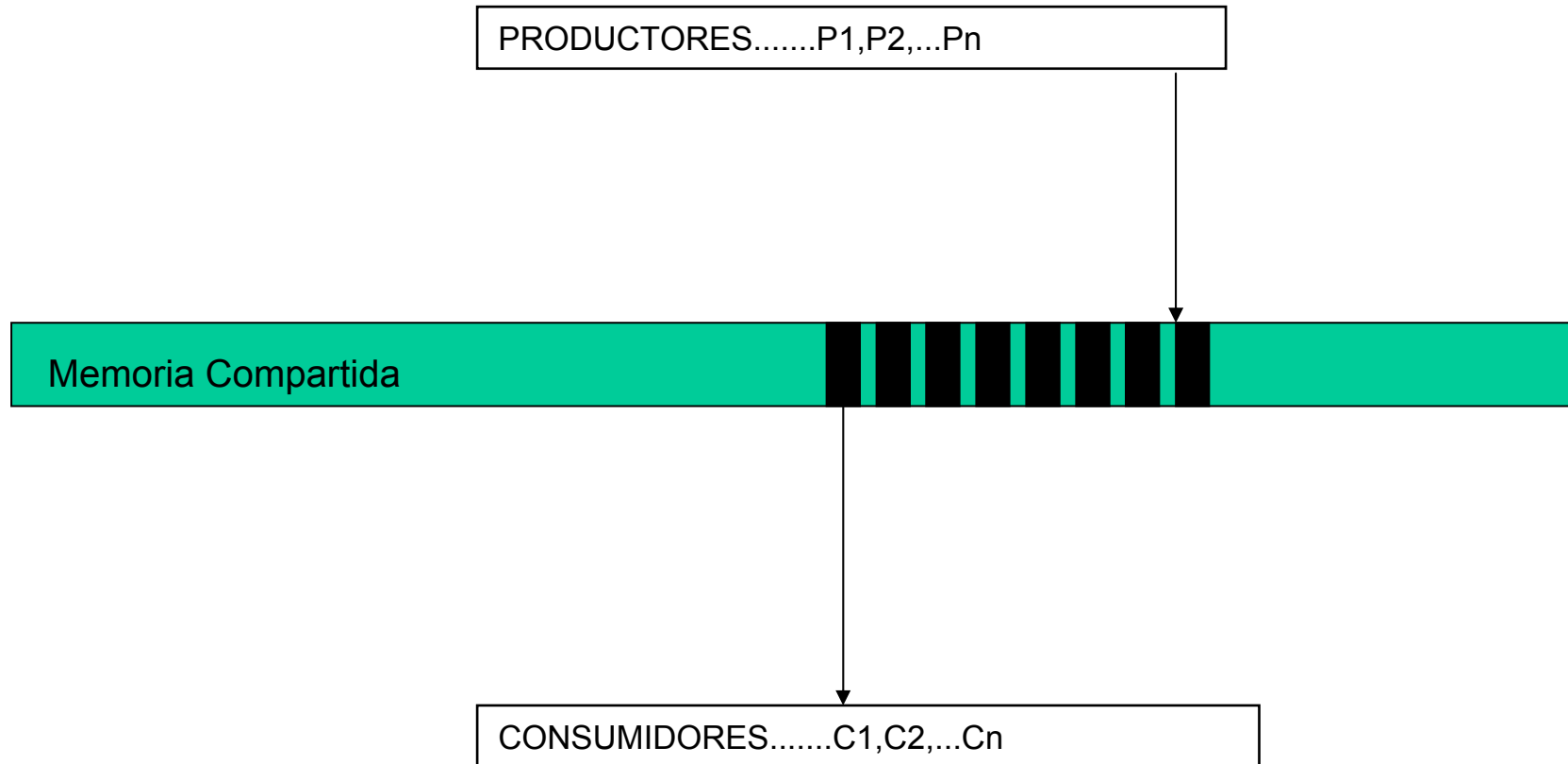
# JBACI

- Estos son los procedimientos para dibujar y editar las figuras:
  - `create(handle, figure, color, x, y, size1, size2)`: Crea un nuevo gráfico con un específico color, localización y tamaño.
  - `changecolor(handle, color)`: Cambia el color de la figura.
  - `makevisible(handle, flag)`: Muestra u oculta su figura dependiendo de si el valor que le pasamos es 0 o 1).
  - `moveto(handle, x, y)`: Mueve la figura a una nueva posición, especificada por nosotros.
  - `moveby(handle, deltax, deltay)`: Mueve la figura a una nueva posición relativa a la posición actual de la figura.

# Ejemplo productores / consumidores

## Descripción Teórica

- Uno o mas productores generan cierto tipo de datos y los sitúan en una zona de memoria o buffer. Un único consumidor saca elementos del buffer de uno en uno. El sistema debe Impedir la superposición de operaciones sobre el buffer



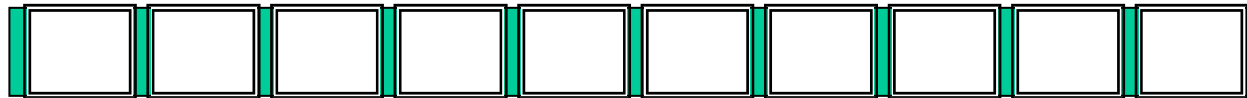
## Aplicación Práctica

**PRODUCTOR**

Generará número de 100 en 100. Estos número se colocarán en en un vector de enteros de tamaño 10.

-Podrá haber un máximo de 10 números distintos en el buffer

-El vector de enteros será el recurso compartido.



N1.....N2.....N3.....N4.....N5.....N6.....N7.....N8.....N9.....N10

**CONSUMIDO**

R

El consumidor tomará los números del buffer y los mostrará en pantalla

jBACI Concurrency Simulator V1.4.5

File Editor Program Options Window Help

Open Save Copy Cut Paste Find Find again Edit Compile Run Go Pause Step Source Step Pcod

```
//Recursos compartidos
1 const int TAMA=10;           //tamaño del buffer compartido
2 const int MAX_CICLOS=1;      //veces que se producen los 100 números
3 int buffer[TAMA];           //buffer compartido
4 semaphore s;                 //semaforo para el control de procesos
5 semaphore vacio;             //semaforo para el control del buffer vacío
6 semaphore lleno;             // semaforo para el control del buffer lleno
7
8 void Productor()
9 {
10     int valor,count;
11     int indice=0;
12     int ciclo_max=0;
13     while (ciclo_max<MAX_CICLOS)
14     {
15         for (cont=0;cont<100;cont++)
16         {
17             //Producimos los números
18             valor=valor++;
19             cout << "Valor producido: " << valor <<endl;
20             wait (lleno);
21             wait(s);
22             //Añadimos
23             buffer[indice]=valor;
24             signal(s);
25             signal(vacio);
26             indice=(indice+1) % TAMA;
27         }
28         ciclo_max=ciclo_max++;
29     }
30 }
```

```
#define tamaño_buffer N
TSemaforo e,s,n;

Void productor()
{
    while (true)
    {
        producir();
        P(e);
        P(s);
        añadir_buffer();
        V(s);
        V(n);
    }
}
```



jBACI Concurrency Simulator V1.4.5

File Editor Program Options Window Help

Open Save Copy Cut Paste Find Find again Edit Compile Run Go Pause

```
32
33 void Consumidor()
34 {
35     int valor, cont;
36     int indice =0;
37     int ciclo_max=0;
38
39     while (ciclo_max < MAX_CICLOS)
40     {
41         for (cont=0; cont<100;cont++)
42         {
43             wait(vacio);
44             wait(s);
45             //Tomar
46             valor=buffer[indice];
47             signal(s);
48             signal(lleno);
49             //Consumir
50             cout << "Elemento consumido: " << valor << endl;
51             indice = (indice +1) %TAMA;
52         }
53         ciclo_max=ciclo_max++;
54     }
55 }
```

```
void consumidor()
{
    while (true)
    {
        P(n);
        P(s);
        coger_buffer();
        V(s);
        V(e);
        consumir();
    }
}
```

jBACI Concurrency Simulator V1.4.5

File Editor Program Options Window Help

Open Save Copy Cut Paste Find Find again Edit Compile Run Go

```
61
62 void main()
63 {
64     initialisesem(s,1);
65     initialisesem(vacio,0);
66     initialisesem(lleno,TAMA);
67     cobegin
68     {
69         Productor();
70         Consumidor();
71     }
72 }
73
74
75
76
77
78
79
```

```
void main()
{
    inicializar(s,1);
    inicializar(n,0);
    inicializar(e,tamaño_buffer);

    cobegin
        productor();
        consumidor();
    coend;
}
```