# 6620. Organización de Computadoras
# Trabajo Práctico 0:
# Infraestructura Básica

González, Juan Manuel, *Padrón Nro. 79.979*
juan0511@yahoo.com

Smocovich, Elizabeth, *Padrón Nro. 86.326*
liz.smocovich@gmail.com

Pereira, Maria Florencia, *Padrón Nro. 88.816*
mflorenciapereira@gmail.com

1er. Cuatrimestre de 2010
66.20 Organización de Computadoras – Prática Jueves
Facultad de Ingeniería, Universidad de Buenos Aires

**Resumen**

El presente trabajo tiene como objetivo familiarizarse con las herramientas de software que serán usadas en los siguientes trabajos, implementando un programa (y su correspondiente documentación) que resuelva el problema piloto planteado, una versión en ANSI C del comando *dc* de UNIX, que implementa una calculadora en notación polaca inversa. A su vez, se utilizará el programa GXemul para simular un entorno de desarrollo en una máquina MIPS (corriendo una reciente versión del sistema operativo NetBSD), a fin de obtener el código en Assembly MIPS32 de la implementación realizada.

# 1. Introducción

Al comenzar a utilizar nuevas herramientas, en cualquier ámbito, es necesaria una breve introduccion al funcionamiento de las mismas: tener una nocioń de las prestaciones que ofrecen, asi también como de sus limitaciones.
Como primer objetivo en la materia, nos proponemos adentrarnos en el funcionamiento del emulador GXemul. Nuestra meta será emular una plataforma MIPS DECstation 5000/200 (ejecutando un sistema operativo NetBSD), para poder desde allí desarrollar programas en ANSI C. Estos serán compilados y ejecutados haciendo uso de la herramienta GCC (compilador C GNU), mediante el cual también será posible obtener, a posteriori, el código MIPS32 del programa.
Como último punto, aprenderemos los rudimientos de LaTeX para generar la documentación relevante al trabajo práctico.

# 2. Programa a implementar

Se trata de una una versión en ANSI C del comando *dc* de UNIX, que implementa una calculadora en notación polaca inversa.
La entrada del programa esta especificada en uno o más archivos. En caso de no especificarse uno, el programa leerá comandos de *stdin*. Debe notarse también que el archivo de nombre '-' representa stdin. El programa escribe la salida en *stdout*. Los mensajes de error deben indicarse via *stderr*.
A continuación se describen los comandos disponibles:

| Comando | Descripción |
|---|---|
| f | Vacia el contenido de la pila, imprimiendo sus contenidos. |
| p | Imprime el valor del tope de la pila, sin alterar la pila. Agrega un caracter de fin de linea. |
| n | Toma el valor del tope de la pila y lo imprime, sin agregar un caracter de fin de linea. |
| + - * % / | Toma los 2 valores en el tope de la pila y empuja el resultado de la operacion. |
| v | Calcula el valor de la raiz cuadrada del elemento en el tope de la pila. |
| c | Vacia la pila. |
| d | Duplica el valor en la cima de la pila. |
| r | Invierte el orden del primer y segundo elemento de la pila. |

# 3. Consideraciones sobre el desarrollo

A continuación se detallan las consideraciones tenidas en cuenta para el desarrollo del trabajo práctico. Hemos decidido separar las que surgen del Diseño de las que surgen en la implementación.

## 3.1. Consideraciones de Diseño

El programa a desarrollar consta a grandes rasgos de un intérprete de operaciones y una estructura LIFO donde son almacenados los operandos y los resultados de las operaciones efectuadas. La pila ha sido implementada como un TDA, y se encuentra codificada en un conjunto de archivos independientes. El resto de las funciones utilizadas se concentran en un solo archivo, ya que se consideró innecesaria una mayor modularización, en base a la baja complejidad y extensión de los algoritmos implementados.

El procesamiento e interpretación de los comandos ingresados desde la consola es manejado mediante la biblioteca 'getopt', evitando así la implementación de un parser para este fin. En cuanto a los datos ingresados en sí, la aplicación ha sido diseñada para que el interprete de operaciones reciba siempre una única cadena de caracteres con toda la información a procesar. En caso de que se especifiquen varios archivos como entrada, primero se concatenan y luego se llama al intérprete con una sola cadena de datos. La salvedad es cuando se opera directamente tomando *stdin* como entrada, ya que en este modo el ingreso y procesamiento de datos es interactivo. Para poder resolver esto, se generó una función que es invocada solamente en este caso y va enviando al intérprete las lineas que el usuario ingresa. Como no se sabe a priori la cantidad de datos que se ingresarán, la reserva de memoria es realizada en forma dinámica.
Finalmente, cabe destacar que todas las impresiones del programa son ruteadas por *stdout*, y los mensajes de error a través de *stderr*.

Los códigos que el programa devuelve al SO son los siguientes:

0: Ejecución sin problemas.
1: Error de ejecución.

## 3.2. Consideraciones de Implementación

Cabe destacar que para implementar el trabajo práctico se partió de la totalidad del programa implementado en lenguage C (ANSI) y luego se procedió a traducir a assembly de MIPS el código final, con el uso de GCC.

### 3.2.1. Portabilidad de la solución

El programa está diseñado para poder ser ejecutado en diferentes plataformas, como por ejemplo NetBSD (PMAX) y la versión de Linux usada para correr el simulador, GNU/Linux (i686). El haber codificado el proyecto en leguage ANSI C garantiza que pueda ser compilado en ambas plataformas sin problemas, dotando al programa de un grado mínimo de portabilidad.
Por otro lado, se destaca que se entrega junto con el presente informe dos versiones del código fuente, una en formato C y la otra en código assembly MIPS.

### 3.2.2. Descripción del la pila utilizada

Para implementar la pila que utiliza la calculadora desarrollada se ha utilizado una estructura LIFO dinámica, almacenada completamente en memoria, utilizando punteros para vincular los distintos nodos. Se ha también dotado a esta implementación la posibilidad de trabajar con cualquier tipo de datos, siendo necesario definir el dato a utilizar en el código y pasando a las primitivas el tamaño del dato definido. Para este trabajo práctico se trabaja con datos del tipo double como elementos de la pila.
Se aclara que el diseño de esta estructura se ha basado en el utilizado para la materia 'Algoritmos y Programación II' de esta facultad, aunque se han realizado diversas modificaciones a ese diseñõ, notablemente la reducción del set de

primitivas a las de creación, push y pop. Todas las validaciones que es necesario realizar a la hora de operar con la estructura han sido incluidas en estas dos últimas primitivas, lo que consideramos simplifica notablemente su operación.

### 3.2.3. Descripción del algoritmo de ejecución de comandos

El algoritmo desarrollado es muy simple, recibe como parámetros una cadena de caracteres y la pila de la calculadora. En la primera de las estructuras mencionadas se encuentran almacenados todos los datos que se han ingresado al programa, independientemente de cual haya sido el método de entrada utilizado. Para llevar a cabo su cometido, la función se encarga de recorrer esta estructura posición por posición y validar los datos allí presentes. Si se trata de un valor numérico, el mismo se inserta en la pila, caso contrario se valida que el caracter detectado sea una operación o comando válido. En caso de ser válido se reliza la operación correspondiente, y en caso contrario se emite por *stderr* un mensaje de error, de manera similar al comando *dc* original.

# 4. Generación de ejecutables y código assembly

Para generar el ejecutable del programa, debe correrse la siguiente sentencia en una terminal:

```
$ gcc -Wall -O0 -o tp0 tp0.c stack.h stack.c
```

Para generar el código MIPS32, debe ejecutarse lo siguiente:

```
$ gcc -Wall -O0 -S -mrnames tp0.c stack.h stack.c
```

Nótese que para ambos casos se han deshaibilitado las optimizaciones del compilador (-O0) y se han activado todos los mensajes de 'Warning' (-Wall). Además, para el caso de MIPS, se han habilitado otras dos banderas, '-S' que detiene al compilador luego de generar el assembly y '-mrnames' que tiene como objetivo generar la salida utilizando los nombres de los registros en vez de los números de registros.

# 5. Corridas de pruebas

En esta seccion se presentan algunas de las distintas corridas que se realizaron para probar el funcionamiento del trabajo práctico.

En primer lugar se mostró el mensaje de ayuda:

```
$ ./tp0 -h
Uso:
  tp0 -h
  tp0 -V
  tp0 [options]
Options:
  -V,  --version               Version.
  -h,  --help                  Ayuda.
  -f,  --scriptfile scrptfile  Ejecuta los comandos especificados en el scriptfile.
  -e,  --script script         Ejecuta los comandos en el script.
Examples:
  tp0
  tp0 -e"22+5*p"
  tp0 -f calculo.m
```

Luego se imprimió la versión del programa:

```
$ ./tp0 -V
TP0: 1.0
```

A continuación se muestran 8 ejemplos mostrando las diversas formas que existen para ejecutar el programa, y la salida obtenida para cada uno de estos casos.

```
$echo "4 2 + 3*p" | ./tp0
18

$cat tmp1 _2 1 +
$cat tmp2 p 3 4 *p
$./tp0 --scriptfile tmp1 tmp2
-1
12

$./tp0 -e "1 2 + 3 + 4+ p"
10

$./tp0 -e "1 2 3f"
3
2
1

$./tp0 -e "1 2 + c 4"

$./tp0 -e "1 2 df"
```

```
2
2
1

$./tp0 -e "p"
ERROR: Pila vacia.

$ ./tp0
2 3 4
+
t
ERRROR: 't' no implementado.
f
7.00000000
2.00000000
f
ERROR: Pila vacia.
```

# 6. Código ANSI C

A continuación se incluye el código fuente del programa.

# 7. Código MIPS32

A continuación se incluye el código MIPS32 del programa.

# 8. Enunciado

A continuación se incluye el enunciado original del trábajo práctico.

# 9. Conclusiones

Como conclusión, podemos considerar que hemos logrado obtener un buen manejo de las herramientas introducidas en este primer proyecto.

## Referencias


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

[2] Oetiker, Tobias, "The Not So Short Introduction To LaTeX2", http://www.physics.udel.edu/~dubois/lshort2e/

[3] dc (Computer Program), http://en.wikipedia.org/wiki/Dc_(computer_program)

[4] Kernighan, Brian W. / Ritchie, Dennis M. El Lenguaje De Prorgamación C. Segunda Edición, PRENTICE-HALL HISPANOAMERICANA SA, 1991.