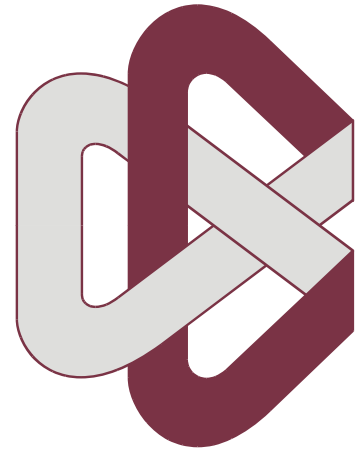

CIMAT

# Optimización de código 2

*Miguel Vargas-Félix*

miguelvargas@cimat.mx
http://www.cimat.mx/~miguelvargas

# Debug vs. release

En **cómputo de alto rendimiento** es importante distinguir los objetivos del código generado en modo debug o release.

La idea es que una vez que el código está probado en modo debug desactivar las verificaciones.

| En el modo debug se requiere: | En modo release se requiere: |
|---|---|
| • Buscar fallas en la lógica del programa. | • **No** buscar fallas en las lógica del programa. |
| • Verificar de límites de valores. | • **No** verificar límites de valores. |
| • Imprimir mensajes de debug. | • **No** imprimir mensajes de debug. |
| • Insertar información para el debugger (en GCC se hace con **-g**). | • **No** insertar información para el debugger. |
| • Detectar errores catastróficos (falta al reservar memoria, falla en acceso a disco, etc). | • Detectar errores catastróficos (falta al reservar memoria, falla en acceso a disco, etc).<br>• Que se ejecute lo más rápidamente posible.<br>• Compilar habilitando la optimización del compilador (en GCC con **-O3**). |

# assert.h

La forma estandar de hacer verificaciones que sólo existan en modo debug es por medio de la función **assert** y del macro **NDEBUG**.

La función **assert** permite verificar condiciones dentro del código. Por ejemplo, en el siguiente código se puede asegurar que no habrá división entre cero.

```
#include <assert.h>

int main() {
    double a = 1;
    double b = 0;

    assert(b != 0);
    double c = a/b;
}
```

En caso de que no se cumpla la condición el programa terminará y se generará un mensaje de error.

Debido a que el programa termina **assert** no debe usarse para verificar entradas del usuario.

La función **assert** debe usarse para verificar la lógica del programa o para buscar bugs.

Una carácterística importante de la función **assert** es que puede desactivarse en el momento de la compilación. Esto se hace definiendo el macro **NDEBUG**.

```
#include <assert.h>

int main() {
    double a = 1;
    double b = 0;

    assert(b != 0);
    double c = a/b;
}
```

```
g++ -o example1 example1.cpp
```

```
#include <assert.h>

int main() {
    double a = 1;
    double b = 0;

    assert(b != 0);
    double c = a/b;
}
```

```
./example1
```

```
g++ -D NDEBUG -o example1 example1.cpp
```

```
#include <assert.h>

int main() {
    double a = 1;
    double b = 0;

    ((void) 0);
    double c = a/b;
}
```

```
./example1
```

Se puede además utilizar el macro NDEBUG para identificar cuando se está compilando en modo debug o en modo release.

```
#include <stdio.h>

int main() {
    printf("Test 1\n");

#if !defined(NDEBUG)
    printf("Test 2\n");
#endif

    printf("Test 3\n");
}
```

```
g++ -o example2 example2.cpp
```

```
#include <stdio.h>

int main() {
    printf("Test 1\n");

    printf("Test 2\n");

    printf("Test 3\n");
}
```

```
./example2
```

```
g++ -D NDEBUG -o example2 example2.cpp
```

```
#include <stdio.h>

int main() {
    printf("Test 1\n");



    printf("Test 3\n");
}
```

```
./example2
```

# Ejemplo con matrices

Por ejemplo, para llenar una matriz

```
#include <assert.h>

#define ROWS 10000000
#define COLS 100

void Set(char* matrix, int i, int j, char value) {
    assert((i >= 0) && (i < ROWS));
    assert((j >= 0) && (j < COLS));
    matrix[i*COLS + j] = value;
}

int main() {
    char* matrix = new char[ROWS*COLS];

    for (int i = 0; i < ROWS; ++i)
        for (int j = 0; j < COLS; ++j)
            Set(matrix, i, j, 0);

    delete [] matrix;
    return 0;
}
```

```
g++ -o example3 example3.cpp
time ./example3
```

```
g++ -D NDEBUG -o example3 example3.cpp
time ./example3
```

El siguiente es un ejemplo de como hacer un makefile que sirva para debug o release.

```
#.SILENT:
.PHONY: release debug clean

CXX=g++
RM=rm -f
STRIP=strip
OUTPUT=example3
OBJS=example3.o
RELEASE_CPPFLAGS=-Wall -Wextra -pedantic -DNDEBUG
RELEASE_CXXFLAGS=-flto -O3
RELEASE_LDFLAGS=-flto
DEBUG_CPPFLAGS=-Wall -Wextra -pedantic
DEBUG_CXXFLAGS=-g
DEBUG_LDFLAGS=

release: CPPFLAGS=$(RELEASE_CPPFLAGS)
release: CXXFLAGS=$(RELEASE_CXXFLAGS)
release: LDFLAGS=$(RELEASE_LDFLAGS)
release: $(OUTPUT)
	$(STRIP) $(OUTPUT)

debug: CPPFLAGS=$(DEBUG_CPPFLAGS)
debug: CXXFLAGS=$(DEBUG_CXXFLAGS)
debug: LDFLAGS=$(DEBUG_LDFLAGS)
debug: $(OUTPUT)

clean:
	$(RM) $(OUTPUT) $(OBJS)

$(OUTPUT): $(OBJS)
	$(CXX) -o $@ $(OBJS) $(LDFLAGS)

example3.o: example3.cpp
	$(CXX) $(CPPFLAGS) $(CXXFLAGS) -o $@ -c $<
```

```
make debug
time ./example3
```

```
make release
time ./example3
```

# ¿Preguntas?

miguelvargas@cimat.mx