Debugging

25 de Agosto de 2023

¿Qué vamos a ver?

- Buenas prácticas

¿Qué vamos a ver?

- Buenas prácticas
- Testing

¿Qué vamos a ver?

- Buenas prácticas
- Testing
- Debugging

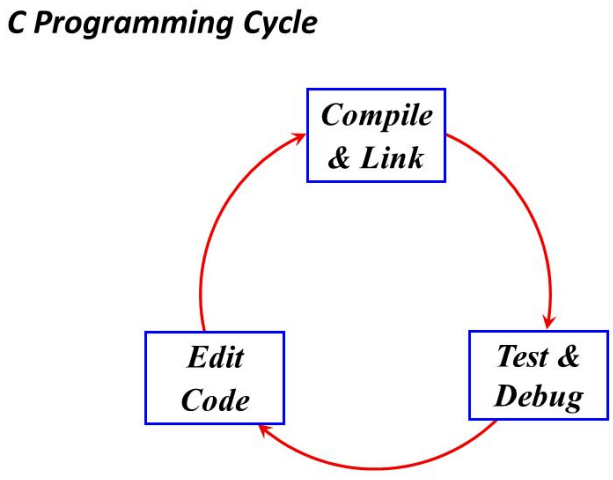
¿Qué vamos a ver?

- Buenas prácticas
- Testing
- Debugging
- Intro al taller

Flujo de trabajo

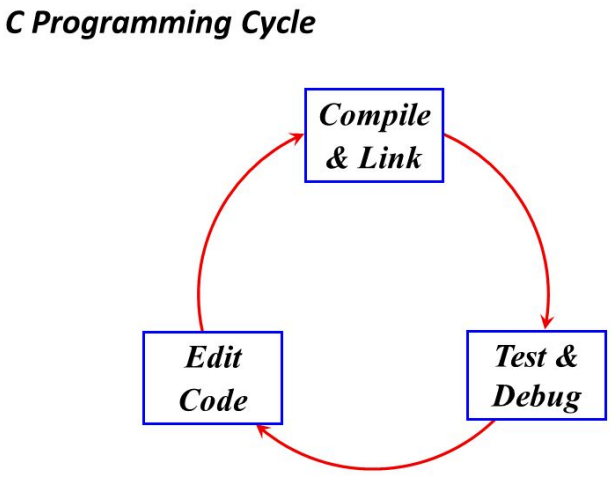
Flujo de trabajo

- Analizo la especificación.



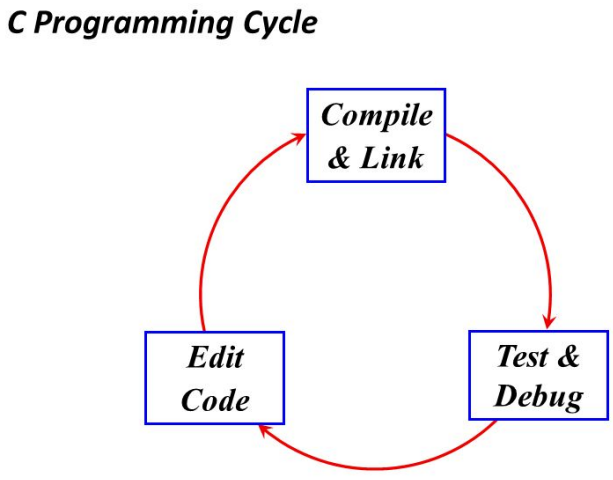
Flujo de trabajo

- Analizo la especificación.
- Codeo



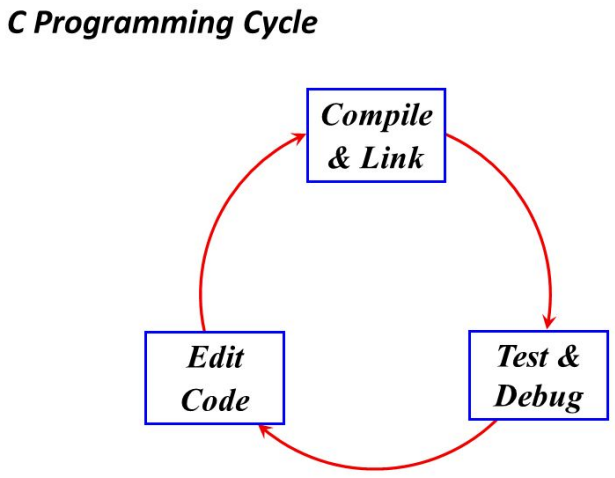
Flujo de trabajo

- Analizo la especificación.
- Codeo
- Compilo



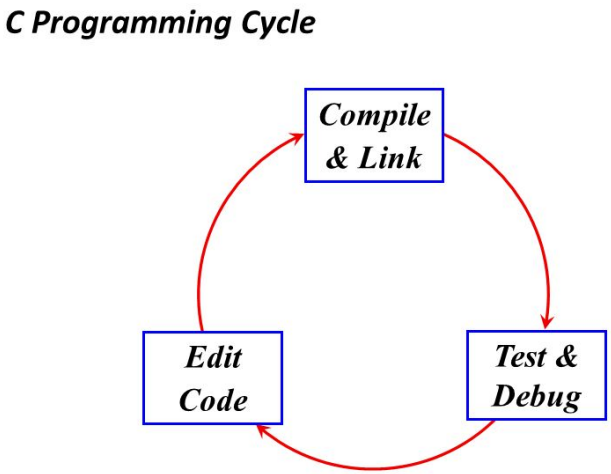
Flujo de trabajo

- Analizo la especificación.
- Codeo
- Compilo
- Testeo



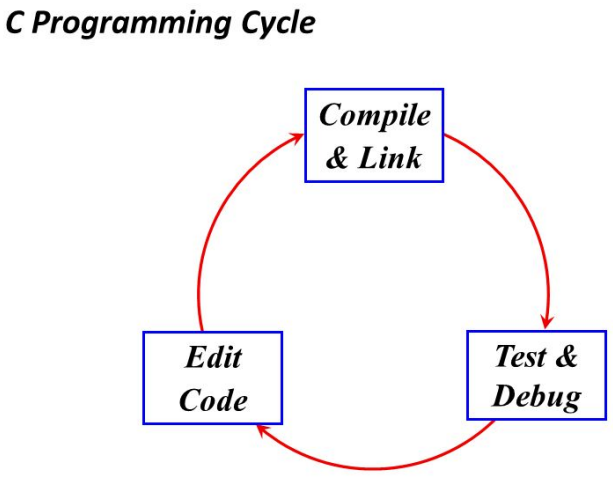
Flujo de trabajo

- Analizo la especificación.
- Codeo
- Compilo
- Testeo
- Debugueo



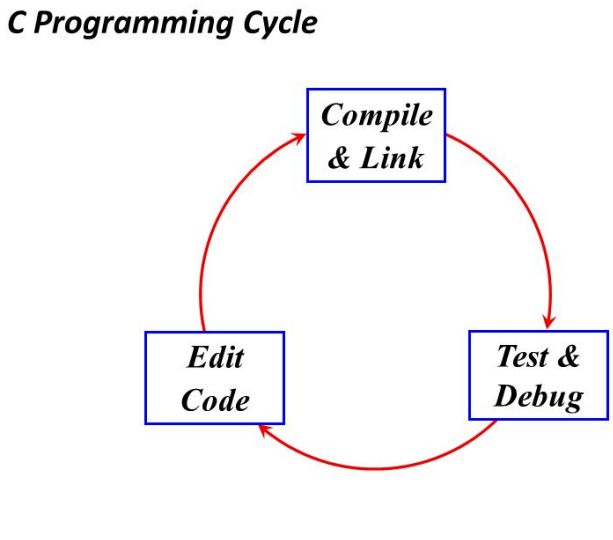
Flujo de trabajo

- Analizo la especificación.
- Codeo
- Compilo
- Testeo
- Debugueo
- Refactorizo



Flujo de trabajo

- Analizo la especificación.
- Codeo
- Compilo
- Testeo
- Debugueo
- Refactorizo
- ... Más tests



Buenas prácticas de programación - Legibilidad

Qué opinan ?

```
for (long l = 4946144450195624l; l > 0; l >>= 5)
        System.out.print((char) (((l & 31 | 64) % 95) + 32));
```

Buenas prácticas de programación - Legibilidad

Qué opinan ?

```
for (long l = 4946144450195624l; l > 0; l >>= 5)
        System.out.print((char) (((l & 31 | 64) % 95) + 32));
```

Priorizar la legibilidad

Buenas prácticas de programación

```
Matriz prod(Matriz A, Matriz B) {
      Matriz C = crear_matriz(filas(A), columnas(B));
      for (int fi=0; fi < filas(A); fi++)
      for (int co = 0; co < columnas(B); fi++)
    C[fi][co] += prod_aux(A[fi], B, co); return C;
    }
    double prod_aux(Fila A, Matriz B, int co) {
    double d = 0;
    for (int i = 0; i < dim(A); ++i)
d = A[i] * B[i][co];  return d;
}
```

Qué opinan ?

Buenas prácticas de programación

```
Matriz prod(Matriz A, Matriz B) {
      Matriz C = crear_matriz(filas(A), columnas(B));
      for (int fi=0; fi < filas(A); fi++)
      for (int co = 0; co < columnas(B); fi++)
    C[fi][co] += prod_aux(A[fi], B, co); return C;
    }
    double prod_aux(Fila A, Matriz B, int co) {
    double d = 0;
    for (int i = 0; i < dim(A); ++i)
d = A[i] * B[i][co];  return d;
}
```

Qué opinan ?

Buenas prácticas de programación

```
Matriz producto(Matriz A, Matriz B) {
    Matriz C = crear_matriz(filas(A), columnas(B));
    for (int fi=0; fi < filas(A); fi++) {
        for (int co = 0; co < columnas(B); fi++) {
            C[fi][co] += producto_fila_columna(A[fi], B, co);
        }
    }
    return C;
}

double producto_fila_columna(Fila A, Matriz B, int co) {
    double d = 0;
    for (int i = 0; i < dim(A); ++i) {
        d = A[i] * B[i][co];
    }
    return d;
}
```

Buenas prácticas de programación

```
Matriz producto(Matriz A, Matriz B) {
    Matriz C = crear_matriz(filas(A), columnas(B));
    for (int fi=0; fi < filas(A); fi++) {
        for (int co = 0; co < columnas(B); fi++) {
            C[fi][co] += producto_fila_columna(A[fi], B, co);
        }
    }
    return C;
}

double producto_fila_columna(Fila A, Matriz B, int co) {
    double d = 0;
    for (int i = 0; i < dim(A); ++i) {
        d = A[i] * B[i][co];
    }
    return d;
}
```

Indentar correctamente

Buenas prácticas de programación

```
Matriz producto(Matriz A, Matriz B) {
    Matriz C = crear_matriz(filas(A), columnas(B));
    for (int fi=0; fi < filas(A); fi++) {
        for (int co = 0; co < columnas(B); fi++) {
            C[fi][co] += producto_fila_columna(A[fi], B, co);
        }
    }
    return C;
}

double producto_fila_columna(Fila A, Matriz B, int co) {
    double d = 0;
    for (int i = 0; i < dim(A); ++i) {
        d = A[i] * B[i][co];
    }
    return d;
}
```

Indentar correctamente

Utilizar nombres adecuados

Buenas prácticas de programación

```
Matriz producto(Matriz A, Matriz B) {
    Matriz C = crear_matriz(filas(A), columnas(B));
    for (int fi=0; fi < filas(A); fi++) {
        for (int co = 0; co < columnas(B); fi++) {
            C[fi][co] += producto_fila_columna(A[fi], B, co);
        }
    }
    return C;
}

double producto_fila_columna(Fila A, Matriz B, int co) {
    double d = 0;
    for (int i = 0; i < dim(A); ++i) {
        d = A[i] * B[i][co];
    }
    return d;
}
```

Indentar correctamente
Utilizar nombres adecuados
Utilizar comentarios adecuados (A veces el buen código se autocomenta)

Buenas prácticas de programación - Otras/Resúmen

- Priorizar la legibilidad.

Buenas prácticas de programación - Otras/Resúmen

- Priorizar la legibilidad.
- Indentación correcta.

Buenas prácticas de programación - Otras/Resúmen

- Priorizar la legibilidad.
- Indentación correcta.
- Coloca comentarios.

Buenas prácticas de programación - Otras/Resúmen

- Priorizar la legibilidad.
- Indentación correcta.
- Coloca comentarios.
- Testea tu código incrementalmente.

Buenas prácticas de programación - Otras/Resúmen

- Priorizar la legibilidad.
- Indentación correcta.
- Coloca comentarios.
- Testea tu código incrementalmente.
- Simplifica al máximo.

Buenas prácticas de programación - Otras/Resúmen

- Priorizar la legibilidad.
- Indentación correcta.
- Coloca comentarios.
- Testea tu código incrementalmente.
- Simplifica al máximo.
- No reproduzcas fragmentos idénticos de código.

Buenas prácticas de programación - Otras/Resúmen

- Priorizar la legibilidad.
- Indentación correcta.
- Coloca comentarios.
- Testea tu código incrementalmente.
- Simplifica al máximo.
- No reproduzcas fragmentos idénticos de código.
- Que una sola función no realice más de 1 funcionalidad. Evitar efectos colaterales.

Buenas prácticas de programación - Otras/Resúmen

- Priorizar la legibilidad.
- Indentación correcta.
- Coloca comentarios.
- Testea tu código incrementalmente.
- Simplifica al máximo.
- No reproduzcas fragmentos idénticos de código.
- Que una sola función no realice más de 1 funcionalidad. Evitar efectos colaterales.
- Que una función no sea de más de 1 página

Buenas prácticas de programación - Otras/Resúmen

- Priorizar la legibilidad.
- Indentación correcta.
- Coloca comentarios.
- Testea tu código incrementalmente.
- Simplifica al máximo.
- No reproduzcas fragmentos idénticos de código.
- Que una sola función no realice más de 1 funcionalidad. Evitar efectos colaterales.
- Que una función no sea de más de 1 página
- Realiza control de versiones. (más en la clase de git...)

Buenas prácticas de programación - Otras/Resúmen

- Priorizar la legibilidad.
- Indentación correcta.
- Coloca comentarios.
- Testea tu código incrementalmente.
- Simplifica al máximo.
- No reproduzcas fragmentos idénticos de código.
- Que una sola función no realice más de 1 funcionalidad. Evitar efectos colaterales.
- Que una función no sea de más de 1 página
- Realiza control de versiones. (más en la clase de git...)
- 80% del costo de un proyecto de software en su ciclo de vida se dedica al mantenimiento.

Testing

¿Para qué sirven los tests?

¿Para qué sirven los tests?

Los tests incrementan nuestra confianza en la correctitud del código testeado.

Etapas de un test (3):

Testing

Etapas de un test (3):

- Preparación
- Ejercitación
- Validación

Testing

- Utilizaremos el framework de Testing JUnit5 (junit.org/junit5)
- ¿Qué línea corresponde a qué etapa del test?

```
1  @Test
2  void testLeerVectorVacio() {
3          Archivos archivos = new Archivos();
4          Scanner entrada = new Scanner("");
5
6          float[] vec = archivos.leerVector(entrada, 0);
7          assertEquals(0, vec.length);
8  }
```

Validación (Assertions)

El framework provee un conjunto de utilidades para realizar validaciones. Alguna de ellas son:

- `assertTrue(boolean condition)`

Validación (Assertions)

El framework provee un conjunto de utilidades para realizar validaciones. Alguna de ellas son:

- `assertTrue(boolean condition)`
- `assertFalse(boolean condition)`

Validación (Assertions)

El framework provee un conjunto de utilidades para realizar validaciones. Alguna de ellas son:

- `assertTrue(boolean condition)`
- `assertFalse(boolean condition)`
- `assertEquals(int expected, Integer actual)`

Validación (Assertions)

El framework provee un conjunto de utilidades para realizar validaciones. Alguna de ellas son:

- `assertTrue(boolean condition)`
- `assertFalse(boolean condition)`
- `assertEquals(int expected, Integer actual)`
- `assertEquals(double expected, double actual, double delta)`

Validación (Assertions)

El framework provee un conjunto de utilidades para realizar validaciones. Alguna de ellas son:

- `assertTrue(boolean condition)`
- `assertFalse(boolean condition)`
- `assertEquals(int expected, Integer actual)`
- `assertEquals(double expected, double actual, double delta)`
- `assertEquals(double expected, double actual, double delta)`

Validación (Assertions)

El framework provee un conjunto de utilidades para realizar validaciones. Alguna de ellas son:

- `assertTrue(boolean condition)`
- `assertFalse(boolean condition)`
- `assertEquals(int expected, Integer actual)`
- `assertEquals(double expected, double actual, double delta)`
- `assertEquals(double expected, double actual, double delta)`
- `assertSame(Object expected, Object actual)`

Validación (Assertions)

El framework provee un conjunto de utilidades para realizar validaciones. Alguna de ellas son:

- `assertTrue(boolean condition)`
- `assertFalse(boolean condition)`
- `assertEquals(int expected, Integer actual)`
- `assertEquals(double expected, double actual, double delta)`
- `assertEquals(double expected, double actual, double delta)`
- `assertSame(Object expected, Object actual)`

Validación (Assertions)

El framework provee un conjunto de utilidades para realizar validaciones. Alguna de ellas son:

- `assertTrue(boolean condition)`
- `assertFalse(boolean condition)`
- `assertEquals(int expected, Integer actual)`
- `assertEquals(double expected, double actual, double delta)`
- `assertEquals(double expected, double actual, double delta)`
- `assertSame(Object expected, Object actual)`

El listado completo se encuentra en https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/ju

- ¿Cuántos tests son suficientes?
- ¿Cómo elijo los valores de entrada a testear?

- ¿Cuántos tests son suficientes?
- ¿Cómo elijo los valores de entrada a testear?

Dos métricas útiles:

- **Cobertura de líneas:** porcentaje de líneas de código cubiertas por los tests.
- **Cobertura de branches:** porcentaje de *branches* (ramificaciones) cubiertas por los tests.

Debugging

Debugging

¿Cómo hago para detectar problemas/errores en mi código?

- Quick & Dirty: Realizar prints en distintos lugares

Debugging

¿Cómo hago para detectar problemas/errores en mi código?

- Quick & Dirty: Realizar prints en distintos lugares
- Ver la pila de llamadas (secuencias de llamadas que llevaron a esa situación)

Debugging

¿Cómo hago para detectar problemas/errores en mi código?

- Quick & Dirty: Realizar prints en distintos lugares
- Ver la pila de llamadas (secuencias de llamadas que llevaron a esa situación)
- Ejecución paso a paso

Debugging

¿Cómo hago para detectar problemas/errores en mi código?

- Quick & Dirty: Realizar prints en distintos lugares
- Ver la pila de llamadas (secuencias de llamadas que llevaron a esa situación)
- Ejecución paso a paso
- Breakpoints

Debugging

¿Cómo hago para detectar problemas/errores en mi código?

- Quick & Dirty: Realizar prints en distintos lugares
- Ver la pila de llamadas (secuencias de llamadas que llevaron a esa situación)
- Ejecución paso a paso
- Breakpoints
- Breakpoints condicionales

Debugging

¿Cómo hago para detectar problemas/errores en mi código?

- Veamos el código de sumatoria

Debugging

¿Cómo hago para detectar problemas/errores en mi código?

- Veamos el código de sumatoria
- Pongamos prints...

Debugging

¿Cómo hago para detectar problemas/errores en mi código?

- Veamos el código de sumatoria
- Pongamos prints...
- Ejecución paso a paso. Hacer step over, luego into.. Cuánto vale sum ? e i ?

Debugging

¿Cómo hago para detectar problemas/errores en mi código?

- Veamos el código de sumatoria
- Pongamos prints...
- Ejecución paso a paso. Hacer step over, luego into.. Cuánto vale sum ? e i ?
- Pongamos un breakpoint en la línea 12...

Debugging

¿Cómo hago para detectar problemas/errores en mi código?

- Veamos el código de sumatoria
- Pongamos prints...
- Ejecución paso a paso. Hacer step over, luego into.. Cuánto vale sum ? e i ?
- Pongamos un breakpoint en la línea 12...
- Pongamos un breakpoint condicional de i == 10.

Ahora sí. A codear!