

Introducción a la Computación (Matemática)

Primer Cuatrimestre de 2018

Brevísima Introducción a la
Organización de Computadoras

Mapa de la materia

- Programas simples en C++. ✓
- Especificación de problemas. ✓
- Correctitud de algoritmos. ✓
- Lenguaje de alto nivel: Python.
- Complejidad algorítmica.
- Problemas clásicos: búsqueda y ordenamiento.
- Recursión algorítmica.
- Tipos abstractos de datos.
- Técnicas algorítmicas.

Mapa de la materia

- Programas simples en C++. ✓
- Especificación de problemas. ✓
- Correctitud de algoritmos. ✓
- Lenguaje de alto nivel: Python.
 - Complejidad algorítmica.
 - Problemas clásicos: búsqueda y ordenamiento.
 - Recursión algorítmica.
 - Tipos abstractos de datos.
 - Técnicas algorítmicas.

Mapa de la materia

- Programas simples en C++. ✓
- Especificación de problemas. ✓
- Correctitud de algoritmos. ✓
- Lenguaje de alto nivel: Python.
- Complejidad algorítmica.
- Problemas clásicos: búsqueda y ordenamiento.
- Recursión algorítmica.
- Tipos abstractos de datos.
- Técnicas algorítmicas.

Mapa de la materia

- Programas simples en C++. ✓
- Especificación de problemas. ✓
- Correctitud de algoritmos. ✓
- Lenguaje de alto nivel: Python.
- Complejidad algorítmica.
- Problemas clásicos: búsqueda y ordenamiento.
- Recursión algorítmica.
- Tipos abstractos de datos.
- Técnicas algorítmicas.

Mapa de la materia

- Programas simples en C++. ✓
- Especificación de problemas. ✓
- Correctitud de algoritmos. ✓
- Lenguaje de alto nivel: Python.
- Complejidad algorítmica.
- Problemas clásicos: búsqueda y ordenamiento.
- Recursión algorítmica.
- Tipos abstractos de datos.
- Técnicas algorítmicas.

Mapa de la materia


- Programas simples en C++. ✓
- Especificación de problemas. ✓
- Correctitud de algoritmos. ✓
- Lenguaje de alto nivel: Python.
- Complejidad algorítmica.
- Problemas clásicos: búsqueda y ordenamiento.
- Recursión algorítmica.
- Tipos abstractos de datos.
- Técnicas algorítmicas.

Mapa de la materia

- Programas simples en C++. ✓
- Especificación de problemas. ✓
- Correctitud de algoritmos. ✓
- Lenguaje de alto nivel: Python.
- Complejidad algorítmica.
- Problemas clásicos: búsqueda y ordenamiento.
- Recursión algorítmica.
- Tipos abstractos de datos.
- Técnicas algorítmicas.


Modelo de cómputo

Modelo teórico en el que se desarrolla la solución a un problema.

- Lápiz y papel 
- Ábaco (2700 AC)
- Máquinas mecánicas
 - Ej: máquina diferencial de Babbage (1822)
- Máquinas de lógica digital (siglo XX)
 - Compuertas lógicas.


Modelo de cómputo

Modelo teórico en el que se desarrolla la solución a un problema.

- Lápiz y papel 
- Ábaco (2700 AC)
- Máquinas mecánicas
 - Ej: máquina diferencial de Babbage (1822)
- Máquinas de lógica digital (siglo XX)
 - Compuertas lógicas.


Modelo de cómputo

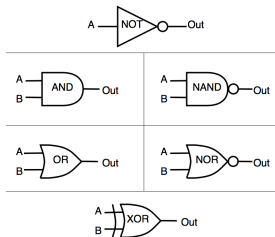
Modelo teórico en el que se desarrolla la solución a un problema.

- Lápiz y papel 
- Ábaco (2700 AC)
- Máquinas mecánicas
 - Ej: máquina diferencial de Babbage (1822)
- Máquinas de lógica digital (siglo XX)
 - Compuertas lógicas.

Modelo de cómputo

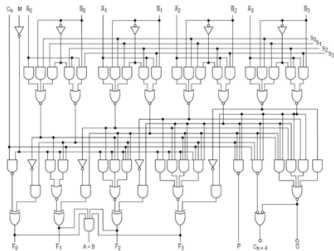
Modelo teórico en el que se desarrolla la solución a un problema.

- Lápiz y papel 
- Ábaco (2700 AC)
- Máquinas mecánicas
 - Ej: máquina diferencial de Babbage (1822)
- Máquinas de lógica digital (siglo XX)
 - Compuertas lógicas.



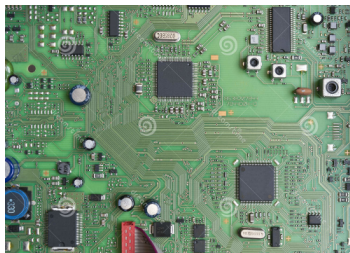
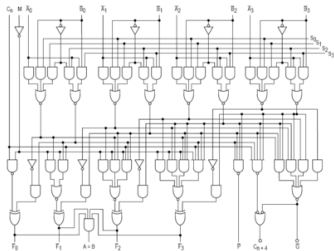
Modelo de cómputo

- Máquinas de lógica digital (siglo XX)
 - Compuertas lógicas: AND, OR, NOT, etc.
 - Programar = construir circuitos electrónicos.
 - Electrodomésticos, relojes, juguetes, etc.



Modelo de cómputo

- Máquinas de lógica digital (siglo XX)
 - Compuertas lógicas: AND, OR, NOT, etc.
 - Programar = construir circuitos electrónicos.
 - Electrodomésticos, relojes, juguetes, etc.



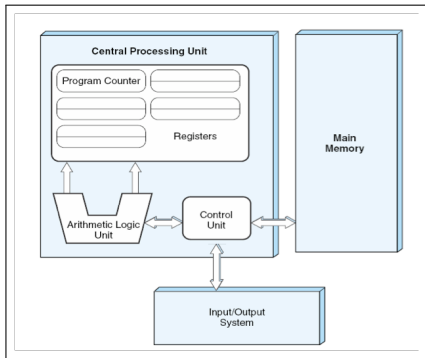
Modelo de cómputo

- Modelo de Von Neumann (computadoras actuales)
 - John Von Neumann (1903-1957): Matemático húngaro.
 - Almacenar los **programas** en la **memoria** de la computadora.
 - Para programar ya no es necesario modificar el hardware.
- **Idea general:** la memoria almacena el programa que resuelve un problema y los datos necesarios para resolverlo.
- La CPU lee y ejecuta el programa, y accede a los datos.

Modelo de cómputo

- Modelo de Von Neumann (computadoras actuales)
 - John Von Neumann (1903-1957): Matemático húngaro.
 - Almacenar los **programas** en la **memoria** de la computadora.
 - Para programar ya no es necesario modificar el hardware.
- **Idea general:** la memoria almacena el programa que resuelve un problema y los datos necesarios para resolverlo.
- La CPU lee y ejecuta el programa, y accede a los datos.

Organización de computadoras



- CPU (Unidad Central de Procesamiento):
 - Unidad de control
 - Unidad aritmético-lógica
 - Registros
- Memoria
- Entrada / Salida
 - Disco rígido, pen drive, mouse, teclado, video, audio, red, wifi, etc.

Ejecución de un programa

Pasos seguidos por la CPU para ejecutar una instrucción:

- 1 Leer la siguiente instrucción a ejecutar.
- 2 Leer de la memoria los datos necesarios para ejecutar la instrucción, y guardarlos en los registros.
- 3 Ejecutar la operación indicada por la instrucción sobre los registros.
- 4 Almacenar el resultado en la posición de memoria que corresponda.

Conjunto de instrucciones de un procesador

- ¿Qué es una **instrucción** para un procesador?
- El procesador tiene un conjunto fijo de instrucciones (lenguaje *assembler*) codificadas en lógica digital.
 - Cada familia de procesadores tiene su propio conjunto de instrucciones.
 - Instrucción = operación simple: suma, asignación, comparación, etc., con operandos simples.
 - Ejemplos: ADD (suma dos registros), INC (incrementa un registro), JMP (salta a otra instrucción), CMP (compara dos registros), JGE (salta a otra instrucción si la última comparación dio \geq), NOP (nada).
 - Conjunto de instrucciones de Intel x86.
- El compilador g++ traduce un programa escrito en C++ a lenguaje de máquina (*assembler*).

Conjunto de instrucciones de un procesador

- ¿Qué es una **instrucción** para un procesador?
- El procesador tiene un conjunto fijo de instrucciones (lenguaje *assembler*) codificadas en lógica digital.
 - Cada familia de procesadores tiene su propio conjunto de instrucciones.
 - Instrucción = operación simple: suma, asignación, comparación, etc., con operandos simples.
 - Ejemplos: ADD (suma dos registros), INC (incrementa un registro), JMP (salta a otra instrucción), CMP (compara dos registros), JGE (salta a otra instrucción si la última comparación dio \geq), NOP (nada).
 - Conjunto de instrucciones de Intel x86.
- El compilador g++ traduce un programa escrito en C++ a lenguaje de máquina (assembler).

Conjunto de instrucciones de un procesador

- ¿Qué es una **instrucción** para un procesador?
- El procesador tiene un conjunto fijo de instrucciones (lenguaje *assembler*) codificadas en lógica digital.
 - Cada familia de procesadores tiene su propio conjunto de instrucciones.
 - Instrucción = operación simple: suma, asignación, comparación, etc., con operandos simples.
 - Ejemplos: ADD (suma dos registros), INC (incrementa un registro), JMP (salta a otra instrucción), CMP (compara dos registros), JGE (salta a otra instrucción si la última comparación dio \geq), NOP (nada).
 - Conjunto de instrucciones de Intel x86.
- El compilador g++ traduce un programa escrito en C++ a lenguaje de máquina (assembler).

Conjunto de instrucciones de un procesador

- ¿Qué es una **instrucción** para un procesador?
- El procesador tiene un conjunto fijo de instrucciones (lenguaje *assembler*) codificadas en lógica digital.
 - Cada familia de procesadores tiene su propio conjunto de instrucciones.
 - Instrucción = operación simple: suma, asignación, comparación, etc., con operandos simples.
 - Ejemplos: ADD (suma dos registros), INC (incrementa un registro), JMP (salta a otra instrucción), CMP (compara dos registros), JGE (salta a otra instrucción si la última comparación dio \geq), NOP (nada).
 - **Conjunto de instrucciones de Intel x86.**
- El compilador g++ traduce un programa escrito en C++ a lenguaje de máquina (assembler).

Conjunto de instrucciones de un procesador

- ¿Qué es una **instrucción** para un procesador?
- El procesador tiene un conjunto fijo de instrucciones (lenguaje *assembler*) codificadas en lógica digital.
 - Cada familia de procesadores tiene su propio conjunto de instrucciones.
 - Instrucción = operación simple: suma, asignación, comparación, etc., con operandos simples.
 - Ejemplos: ADD (suma dos registros), INC (incrementa un registro), JMP (salta a otra instrucción), CMP (compara dos registros), JGE (salta a otra instrucción si la última comparación dio \geq), NOP (nada).
 - **Conjunto de instrucciones de Intel x86.**
- El compilador g++ traduce un programa escrito en C++ a lenguaje de máquina (assembler).

Ejemplo de código C++ compilado

```
/*
Para ver el código assembler,
compilar con "g++ -S nombre.cpp".
*/
int main() {
    int a = 111;
    int b = a + 777;
    char c;
    if (b > 10) {
        c = 'x';
    }
    else {
        c = 'y';
    }
    while (a < 1000) {
        a = a + 1;
    }
}
```

```
main:
    pushq   %rbp
    movq    %rsp, %rbp
    movl    $111, -12(%rbp)
    movl    -12(%rbp), %eax
    addl    $777, %eax
    movl    %eax, -8(%rbp)
    cmpl    $10, -8(%rbp)
    jle     .L2
    movb    $120, -1(%rbp)
    jmp     .L4
.L2:
    movb    $121, -1(%rbp)
    jmp     .L4
.L5:
    addl    $1, -12(%rbp)
.L4:
    cmpl    $999, -12(%rbp)
    setle   %al
    testb   %al, %al
    jne     .L5
    movl    $0, %eax
    popq    %rbp
```


Ejemplo de traducción del condicional

```
if (i == 3 && j > 4) {  
    Bloque_A  
else {  
    Bloque_B  
}  
Bloque_C
```

- Supongamos **i** y **j** almacenados en memoria con etiquetas de ese nombre.
- La conjunción se evalúa **por partes**, y en cuanto una condición falla, se salta a la porción de código del **else**.

```
                cmp $3, i  
                jne Bloque_B  
                cmp $4, j  
                jle Bloque_B  
Bloque_A: ...  
                ...  
                jmp Bloque_C  
Bloque_B: ...  
                ...  
Bloque_C: ...  
                ...
```

Lenguajes de bajo y alto nivel

- Casi nadie programa en assembler.
- El lenguaje C++ está muy cerca del lenguaje assembler.
 - Ejemplo: los arreglos en C++ direccionan memoria física.
 - Se puede incluir fragmentos de assembler en medio del código C++, para lograr mayor eficiencia en segmentos críticos de código.
- C++ es un lenguaje de 'bajo nivel'.
- A partir de ahora vamos a trabajar con un lenguaje de 'alto nivel': Python.
 - Otros ejemplos: Java, Perl, Matlab, Ruby, PHP, etc.
- Mayor distancia entre un lenguaje y assembler:
 - ↑ abstracción; ↑ usabilidad; ↓ eficiencia; ↓ confiabilidad.

Lenguajes de bajo y alto nivel

- Casi nadie programa en assembler.
- El lenguaje C++ está muy cerca del lenguaje assembler.
 - Ejemplo: los arreglos en C++ direccionan memoria física.
 - Se puede incluir fragmentos de assembler en medio del código C++, para lograr mayor eficiencia en segmentos críticos de código.
- C++ es un lenguaje de 'bajo nivel'.
- A partir de ahora vamos a trabajar con un lenguaje de 'alto nivel': Python.
 - Otros ejemplos: Java, Perl, Matlab, Ruby, PHP, etc.
- Mayor distancia entre un lenguaje y assembler:
 - ↑ abstracción; ↑ usabilidad; ↓ eficiencia; ↓ confiabilidad.

Lenguajes de bajo y alto nivel

- Casi nadie programa en assembler.
- El lenguaje C++ está muy cerca del lenguaje assembler.
 - Ejemplo: los arreglos en C++ direccionan memoria física.
 - Se puede incluir fragmentos de assembler en medio del código C++, para lograr mayor eficiencia en segmentos críticos de código.
- C++ es un lenguaje de 'bajo nivel'.
- A partir de ahora vamos a trabajar con un lenguaje de 'alto nivel': Python.
 - Otros ejemplos: Java, Perl, Matlab, Ruby, PHP, etc.
- Mayor distancia entre un lenguaje y assembler:
 - ↑ abstracción; ↑ usabilidad; ↓ eficiencia; ↓ confiabilidad.

Sobre los lenguajes de programación...

- Elementos básicos de programación (variable, asignación, condicional, ciclo): son muy similares en todos los lenguajes.
- Estructuras de control especiales: for, loop, repeat, until, for each, elsif, unless, ... ¡Leer documentación!
- Bibliotecas y funciones auxiliares (ejemplos en C++: string.h, stdio.h, stdbool.h). ¡Leer documentación!
- No hay ningún misterio, y en Internet está la respuesta a casi todas las dudas (de programación).

Sobre los lenguajes de programación...

- Elementos básicos de programación (variable, asignación, condicional, ciclo): son muy similares en todos los lenguajes.
- Estructuras de control especiales: for, loop, repeat, until, for each, elsif, unless, ... ¡Leer documentación!
- Bibliotecas y funciones auxiliares (ejemplos en C++: string.h, stdio.h, stdbool.h). ¡Leer documentación!
- No hay ningún misterio, y en Internet está la respuesta a casi todas las dudas (de programación).

Sobre los lenguajes de programación...

- Elementos básicos de programación (variable, asignación, condicional, ciclo): son muy similares en todos los lenguajes.
- Estructuras de control especiales: for, loop, repeat, until, for each, elsif, unless, ... ¡Leer documentación!
- Bibliotecas y funciones auxiliares (ejemplos en C++: string.h, stdio.h, stdbool.h). ¡Leer documentación!
- No hay ningún misterio, y en Internet está la respuesta a casi todas las dudas (de programación).

Sobre los lenguajes de programación...

- Elementos básicos de programación (variable, asignación, condicional, ciclo): son muy similares en todos los lenguajes.
- Estructuras de control especiales: for, loop, repeat, until, for each, elsif, unless, ... ¡Leer documentación!
- Bibliotecas y funciones auxiliares (ejemplos en C++: string.h, stdio.h, stdbool.h). ¡Leer documentación!
- No hay ningún misterio, y en Internet está la respuesta a casi todas las dudas (de programación).

Sobre los lenguajes de programación...

- Elementos básicos de programación (variable, asignación, condicional, ciclo): son muy similares en todos los lenguajes.
- Estructuras de control especiales: for, loop, repeat, until, for each, elsif, unless, ... ¡Leer documentación!
- Bibliotecas y funciones auxiliares (ejemplos en C++: string.h, stdio.h, stdbool.h). ¡Leer documentación!
- No hay ningún misterio, y en Internet está la respuesta a casi todas las dudas (de programación).

```

#include <stdio.h>
#include <stdlib.h>

// Dice si los elementos del arreglo de enteros A (de longitud size)
// estan ordenados en forma estrictamente creciente.
bool creciente(int A[], int size) {
    int i = 0;
    // Avanzo mientras sea creciente.
    while (i < size-1 && A[i] < A[i+1]) {
        i = i + 1;
    }
    // Si llegue al final, es creciente.
    return (i == size-1);
}

int main(int argc, char* argv[]) {
    // Defino el arreglo en forma dinamica.
    int size = argc - 1;
    int* array = new int[size];

    // Leo el arreglo de la linea de comandos.
    int i = 0;
    while (i < size) {
        array[i] = atoi(argv[i + 1]);
        i = i + 1;
    }

    // Imprimo un mensaje diciendo si es creciente o no.
    if (creciente(array, size)) {
        printf("es creciente\n");
    } else {
        printf("no es creciente\n");
    }
}

```

```
#!/usr/bin/python
import sys

# Dice si los elementos del arreglo de enteros A
# estan ordenados en forma estrictamente creciente.
def creciente(A):
    i = 0
    # Avanzo mientras sea creciente.
    while i < len(A)-1 and A[i] < A[i+1]:
        i = i + 1

    # Si llegue al final, es creciente.
    return i == len(A)-1

# Creo un arreglo (o "lista")
array = list()

# Leo el arreglo de la linea de comandos.
i = 0
while i + 1 < len(sys.argv):
    array.append(sys.argv[i + 1])
    i = i + 1

# Imprimo un mensaje diciendo si es creciente o no.
if creciente(array):
    print "es creciente"
else:
    print "no es creciente"
```

```
#!/usr/bin/perl

# Dice si los elementos del arreglo de enteros A
# estan ordenados en forma estrictamente creciente.
sub creciente {
    my @A = @_;
    my $i = 0;
    # Avanzo mientras sea creciente.
    while ($i < @A-1 && $A[$i] < $A[$i+1]) {
        $i = $i + 1;
    }

    # Si llegue al final, es creciente.
    return $i == @A-1;
}

# Creo un arreglo (o "lista")
my @array;

# Leo el arreglo de la linea de comandos.
my $i = 0;
while ($i + 1 < @ARGV) {
    push @array, $ARGV[$i + 1];
    $i = $i + 1;
}

# Imprimo un mensaje diciendo si es creciente o no.
if (creciente(@array)) {
    print "es creciente\n";
} else {
    print "no es creciente\n";
}
}
```

Archivos

- Recursos para almacenar información (ej: textos, fotos) en un medio durable (ej: disco rígido).
- Cada aplicación maneja uno o más **formatos** de archivos:
 - **Texto plano**. Ej: archivos `.txt`, código C++.
 - **Binario**. Ej: ejecutables (`a.out`, `.exe`), fotos (`.gif`), audio (`.mp3`), docs de procesadores de texto (`.doc`), planillas de cálculo (`.xls`), etc.
- **Operaciones**:
 - **Crear** y **eliminar** un archivo.
 - Especificar sus **permisos**: lectura, escritura y ejecución.
 - **Abrir** y **cerrar** un archivo para usar su contenido, en modo **lectura** o **escritura** (*overwrite* vs. *append*).
 - **Ejecutar** un archivo.

Archivos

- Recursos para almacenar información (ej: textos, fotos) en un medio durable (ej: disco rígido).
- Cada aplicación maneja uno o más **formatos** de archivos:
 - **Texto plano**. Ej: archivos .txt, código C++.
 - **Binario**. Ej: ejecutables (a.out, .exe), fotos (.gif), audio (.mp3), docs de procesadores de texto (.doc), planillas de cálculo (.xls), etc.
- Operaciones:
 - **Crear y eliminar** un archivo.
 - Especificar sus **permisos**: lectura, escritura y ejecución.
 - **Abrir y cerrar** un archivo para usar su contenido, en modo **lectura** o **escritura** (*overwrite* vs. *append*).
 - **Ejecutar** un archivo.

Archivos

- Recursos para almacenar información (ej: textos, fotos) en un medio durable (ej: disco rígido).
- Cada aplicación maneja uno o más **formatos** de archivos:
 - **Texto plano**. Ej: archivos .txt, código C++.
 - **Binario**. Ej: ejecutables (a.out, .exe), fotos (.gif), audio (.mp3), docs de procesadores de texto (.doc), planillas de cálculo (.xls), etc.
- **Operaciones:**
 - **Crear** y **eliminar** un archivo.
 - Especificar sus **permisos**: lectura, escritura y ejecución.
 - **Abrir** y **cerrar** un archivo para usar su contenido, en modo **lectura** o **escritura** (*overwrite* vs. *append*).
 - **Ejecutar** un archivo.

Repaso de la clase de hoy

- Modelo de Von Neumann
- Computadora: CPU + memoria + entrada/salida.
- Conjunto de instrucciones del procesador.
- Lenguajes de bajo nivel (C++) y alto nivel (Python).
- Archivos.

Próximas clases

- Hoy: Introducción a Python.
- Complejidad algorítmica.
- Algoritmos de búsqueda.
- Algoritmos de ordenamiento.