Introducción a la Programación

Usando C, C++, JavaScript, Lilly, ...

Consideraciones Preliminares

- Acerca de la enseñanza, relación entre teoría y practica, abstracción gradual, magia, ...
- Leslie Lamport: https://www.youtube.com/watch?v=rkZzg7Vowao

- No soy operador de sistemas, por ti seré
  - Tienen que hacer su mejor esfuerzo para combatir la adversidad

- Evaluación del laboratorio
  - “engagement”: estar “pilas” investigar y resolver problemas (ver punto anterior), participación, ...

Introducción a las herramientas de programación: Git, GitHub, Comandos de Unix, y Compiladores

- Adquisición de conocimiento de las herramientas: interactiva, en clase, y autodidáctica.
  - Sin embargo mini-manuales ("primers") van a ser preparados a lo largo del curso
- Compiladores Oficiales
  - Windows y Linux : g++, gcc, clang
  - En Windows, requieren instalación de MinGW o activación de WSL
- IDE Aceptados para Entregas
  - Windows: Microsoft Visual Studio 2017 y 2022
  - Pero serán retirados para entregas si los estudiantes no logran entregas limpias

Para Comenzar

- Los estudiantes deben referirse a las presentaciones enviadas por correo en este orden:
  1. Herramientas de Programación
  2. Configuración del Ambiente de Desarrollo
  3. Introducción a la Programación (estas láminas)

- Primero instalar las herramientas y configurar su ambiente de desarrollo (1. y 2.)
  - Verifiquen que pueden usar las herramientas, compilar, y ejecutar programas en C/C++

- Segundo repasar el material en esta presentación y hacer las tareas al final

minimal

Programa mínimo en C:

```
main(){}
```

Aunque debería ser:

```
int main()
{
  return 0;
}
```

Códigos de Salida de Unix

- La función main debe (en general) retornar un código de salida
  - El código de salida es un valor entero entre 0 y 255
  - 0 indica que el programa terminó de manera normal
  - 1 – 255 hubo algún problema

- https://shapeshed.com/unix-exit-codes/
- https://www.geeksforgeeks.org/exit-codes-in-c-c-with-examples/

Hello, World!

```
///
/// hello.cxx
///

#include <stdio.h>

int main()
{
  fprintf(stdout, "Hello, world\n");
}
```

Programación Imperativa: modelo “3 cajitas”

Estado inicial vacio
(antes de ejecutar los pasos)

Deliberadamente, este ejemplo no usa la sintaxis del lenguaje C

<- denota el operador de asignación

Definiciones de variables

```
1. int x <- 6;

2. int y <- 7;

3. int z <- x * y;
```

(Re)asignaciones de variables

```
4. y <- y + 1;

5. x <- 2 * z;
```

Definiciones de Variables

- Sintaxis en lenguaje C y derivados: <tipo> <símbolo> = <expresión> ;
- Ejemplos:

```
int i = 0;
double pi = 3.14159265359;
double y = m * x + q;
```

- Semántica (operacional: informal y simplificada) de la definición de una variable

Los ejemplos anteriores muestran tres piezas claves de información:

un tipo T, un identificador S, y una expresión E (al lado derecho del símbolo de asignación =)

Prerrequisito: el tipo T de la variable S debe ser compatible con el tipo de la expresión E

Pasos

1. **Crear** la cajita S – i.e. reservar una área de memoria M para la variable S
2. **Evaluar** la expresión E, obteniendo como resultado un valor V
3. **Asignar** V a la cajita S – i.e. copiar el valor V al área de memoria M

Asignaciones a variables ya definidas

- Sintaxis en lenguaje C y derivados: <lhs> = <rhs> ;
  - donde <lhs> y <rhs> son expresiones de C/C++, o JavaScript, o Java entre otros

- Semántica (operacional: informal y simplificada) de una asignación

  Prerrequisitos:

  <lhs> debe denotar una cajita (variable) - en la jerga PLDI decimos que <lhs> tiene lvalue

  el tipo de <lhs> debe ser consistente con el tipo de <rhs>

  Pasos

  1. **Evaluar** la expresión E, obteniendo como resultado un valor V
  2. **Asignar** V a la cajita S – i.e. copiar el valor V al área de memoria correspondiente a S

Programación Imperativa: modelo 3 cajitas

Estado al completar el paso 1.

x
6

Deliberadamente, este ejemplo no usa la sintaxis del lenguaje C

<- denota el operador de asignación

```
1. int x <- 6;

2. int y <- 7;

3. int z <- x * y;

4. y <- y + 1;

5. x <- 2 * z;
```

Programación Imperativa: modelo 3 cajitas

Estado al completar el paso 2.

x	y
6	7

Deliberadamente, este ejemplo no usa la sintaxis del lenguaje C

<- denota el operador de asignación

```
1. int x <- 6;

2. int y <- 7;

3. int z <- x * y;

4. y <- y + 1;

5. x <- 2 * z;
```

Programación Imperativa: modelo 3 cajitas

	x	y	z
0.			
1.	6		
2.	6	7	
3.			
4.			
5.			

Deliberadamente, este ejemplo no usa la sintaxis del lenguaje C

<- denota el operador de asignación

```
1. int x <- 6;

2. int y <- 7;

3. int z <- x * y;

4. y <- y + 1;

5. x <- 2 * z;
```

Programación Imperativa: modelo 3 cajitas

	x	y	z
0.			
1.	6		
2.	6	7	
3.	6	7	42
4.			
5.			

Deliberadamente, este ejemplo no usa la sintaxis del lenguaje C

<- denota el operador de asignación

```
1. int x <- 6;

2. int y <- 7;

3. int z <- x * y;

4. y <- y + 1;

5. x <- 2 * z;
```

Programación Imperativa: modelo 3 cajitas

	x	y	z
0.			
1.	6		
2.	6	7	
3.	6	7	42
4.	6	8	42
5.			

Deliberadamente, este ejemplo no usa la sintaxis del lenguaje C

<- denota el operador de asignación

```
1. int x <- 6;

2. int y <- 7;

3. int z <- x * y;

4. y <- y + 1;

5. x <- 2 * z;
```

Programación Imperativa: modelo 3 cajitas

	x	y	z
0.			
1.	6		
2.	6	7	
3.	6	7	42
4.	6	8	42
5.	84	8	42

Deliberadamente, este ejemplo no usa la sintaxis del lenguaje C

<- denota el operador de asignación

```
1. int x <- 6;

2. int y <- 7;

3. int z <- x * y;

4. y <- y + 1;

5. x <- 2 * z;
```

Programación Imperativa: modelo 3 cajitas

	x	y	z
0.			
1.	6		
2.	6	7	
3.	6	7	42
4.	6	8	42
5.	84	8	42

El mismo ejemplo usando la sintaxis de C/C++, JavaScript, Java

= denota el operador de asignación

```
1. int x = 6;

2. int y = 7;

3. int z = x * y;

4. y = y + 1;

5. x = 2 * z;
```

Para la Tarea
Estado de la memoria después de ejecutar cada paso

Este es el ejemplo anterior, ya resuelto, para mostrar como pueden reportar el resultado en texto
Simple texto (.txt) en blanco y negro: ¡no desperdicien tiempo poniendo colores!

```
1. int x = 6;          // x:6

2. int y = 7;          // x:6, y:7

3. int z = x * y;      // x:6, y:7, z:42

4. y = y + 1;          // x:6, y:8, z:42

5. x = 2 * z;          // x:84, y:8, z:42
```

Tarea 0:
Reportar el estado de la memoria después de ejecutar cada paso

Caso A

```
1. int x = 1;
2. int z = 0;
3. z = z + x;
4. x = x + 2;
5. z = z + x;
6. x = x + 2;
7. z = z + x;
8. x = x + 2;
9. z = z + x;
10. x = x + 2;
```

Caso B

```
1. int x = 6;
2. int y = 7;
3. int z = x * y;
4. x = x * x + y;
5. y = x * (x - z);
6. z = z + 1;
```

Caso C

```
1. int p = 0;
2. int u = 1;
3. int s = u + p;
4. p = u;
5. u = s;
6. s = u + p;
7. p = u;
8. u = s;
9. s = u + p;
10. p = u;
11. u = s;
12. s = u + p;
```

Tarea 1:
Intercambiar el valor de dos variables

Utilizando solo definiciones de variables y asignaciones, agregue instrucciones a la siguiente secuencia para intercambiar los valores de *x* y *y*. Claro está que su fragmento de código debe ser general, es decir que debe funcionar independientemente de los valores iniciales de *x* y *y*.

```
1. int x = 6;
2. int y = 7;
…. …
```

C/C++: Palabras reservadas para tipos básicos

- Valores booleanos
  - bool
- Caracteres
  - char
- Valores enteros (representables)
  - char, short int, int, long int, long long int
- Números representados en punto flotante
  - float, double, long double

C/C++: Operadores binarios infijos para comparar números

- Estos operadores funcionan para "números enteros" (int) y "números reales" (float, double)
- Hasta ahora solo hemos visto "números enteros" en C y mencionado a ciertos "números reales"
  - int: números enteros, con signo, *dentro de un rango limitado*
  - float: aproximación a "números reales", usando precisión simple con punto flotante (típicamente 32 bits)
  - double: aproximación a "números reales", usando doble precisión con punto flotante (típicamente 64 bits)
- Si el tipo de los operandos es mixto, se aplican reglas de conversión implícitas (promoción) para hacer que sean iguales
  - int ➔ float
  - int ➔ double
  - float ➔ double
- Independientemente de lo anterior, muchos compiladores fuerzan el uso de doble precisión *al evaluar expresiones* con puntos flotantes
  - Lógicamente, la representación float es respetada *al almacenar los resultados* de la evaluación en variables de tipo float

Operador	Significado	Ejemplo
`<`	menor que	`x < y`
`<=`	menor o igual a	`x <= y`
`==`	igual a	`x == y`
`!=`	distinto de	`x != y`
`>=`	mayor o igual a	`x >= y`
`>`	mayor que	`x > y`

C/C++: Especificadores de formatos: para uso con printf, fprintf, sprintf, etc.

Tipo	*Tipo*	Formateador de Salida
Booleano	bool	n.a. (definido por el programador)
Entero	int	%d
Punto flotante	float y double	%f %lf (con consideraciones adicionales)
String	char *	%s

C/C++: Estructuras para Controlar el Flujo Secuencial

- Condicionales:
  - if
  - switch
- Lazos:
  - while
  - do
  - for

C/C++: Instrucción if – if statement

- Sintaxis 1

```
if (<exp>) {
        ... lista de instrucciones a ejecutar si la expresión <exp> se cumple ...
} else {
        ... lista de instrucciones a ejecutar si la expresión <exp> no se cumple ...
}
```

- Sintaxis 2

```
if (<exp>) {
        ... lista de instrucciones a ejecutar si la expresión <exp> se cumple ...
}
```

C/C++: Instrucción while – while statement

- Sintaxis

```
while (<exp>) {
        <body>
}
```

- Semántica

Mientras <exp> se cumple (evalúa a true) ejecutar el cuerpo <body> del while, es decir …

… si <exp> ➔ true, ejecutar <body> y repetir el while, si no, continuar con la instrucción que sigue

Tarea 2:
Computar con Lazos

Como seguramente apreciaron, los casos A y C de la tarea 0 tienen secuencias repetitivas: por lo tanto se prestan a ser mejorados (generalizados y más cortos a la vez) con el uso de lazos.

Completen los fragmentos de código siguientes (simplemente escríbanlos en un archivo de texto) para lograr tal objetivo. El código debe imprimir la secuencia de valores que nos interesa: z para el caso A y s para el caso C, por lo tanto deben incluir una instrucción fprintf en el cuerpo. Hay varias opciones para la condición de ejecución del lazo: por ahora simplemente asegúrense de que la ejecución termina después de imprimir un numero razonable de valores. Bonus: pueden poner su código dentro de la función main y ejecutar el programa de veras, si completaron su ambiente de desarrollo.

Caso A

```
int x = 1;
int z = 0;
while (<condición>) {
   <cuerpo>
}
```

Caso C

```
int p = 0;
int u = 1;
int s = u + p;
while (<condición>) {
   <cuerpo>
}
```

Contando de 1 a 10: uso de variables y lazos

```c
#include <stdio.h>

int main()
{
  int i = 1;
  while (i <= 10) {
    fprintf(stdout, "%d\n", i);
    i = i + 1;
  }
  return 0;
}
```

Pasando argumentos al programa

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{
  fprintf(stdout, "argc: %d\n\n", argc);
  int i = 0;
  while (i < argc) {
    fprintf(stdout, "argv[%d]: %s\n", i, argv[i]);
    i = i + 1;
  }
  return 0;
}
```

Contando de 1 a N, uso de argumentos

```c
// Advertencia: este programa deliberadamente tiene errores y omisiones!

#include <stdio.h>

int main(int argc, const char *argv[])
{
  // obtener y chequear el valor del argumento
  int N = argv[1]; // <<< el compilador debe rep

  // contar de 1 a N
  int i = 1;
  while (i <= N) {
    fprintf(stdout, "%d\n", i);
    i = i + 1;
  }

  return 0;
}
```

Contando de 1 a N, uso de argumentos con chequeo

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, const char *argv[])
{
  // si el número de argumentos es distinto de 2, terminar con un código de error
  // recuerden que el número de argumentos incluye el nombre del ejecutable, almacenado en argv[0]
  if (argc != 2) {
    fprintf(stderr, "uso: %s <int N>\n", argv[0]);
    return 1;
  }

  // obtener y chequear el valor del argumento
  // notar el uso de la función atoi para convertir el string a entero
  int N = atoi(argv[1]);
  // si N es menor o igual a 0, terminar con un código de error
  if (N <= 0) {
    fprintf(stderr, "N debe ser mayor que cero, pero N == %d\n", N);
    return 1;
  }

  // por fin: contar de 1 a N!
  int i = 1;
  while (i <= N) {
    fprintf(stdout, "%d\n", i);
    i = i + 1;
  }
  return 0;
}
```

Ejercicio para la casa: hola con un argumento opcional
Completar este programa

```c
#include <stdio.h>

int main(int argc, const char *argv[])
{
  if (argc == 1) {
    // Simplemente imprimir "Hola!"
  } else if (argc == 2) {
    // Imprimir "Hola <nombre>!", donde <nombre> es pasado en la línea de comando
    // Por ejemplo: "Hola Francisco!"
  } else {
    fprintf(stderr, "uso: %s [<nombre>]\n", argv[0]);
    return 1;
  }
  return 0;
}
```

THIS IS
SPARTA