Apuntes de Programación – Parcial

Bloque 1 – Fundamentos

Variables y tipos de datos

* Una variable es un espacio en memoria donde se guarda un valor.
* Cada variable tiene un tipo de dato, que define qué clase de información puede almacenar.
* Tipos principales:  
  int: números enteros  
  float o double: números decimales  
  char: un solo carácter, se escribe con comillas simples ('A')  
  bool: valores lógicos (true o false)  
  std::string: texto, se escribe entre comillas dobles ("Hola")

Ejemplo:

int edad = 20;

float precio = 4.99;

char inicial = 'J';

bool aprobado = true;

std::string nombre = "Ana";

Entrada y salida de datos

* Para mostrar datos usamos std::cout.
* Para leer datos del usuario usamos std::cin.
* El operador << se usa para sacar datos, y >> para meterlos.

Ejemplo:

int numero;

std::cout << "Ingrese un número: ";

std::cin >> numero;

std::cout << "Ingresó: " << numero << std::endl;

Operadores

* Aritméticos: +, -, \*, /, % (módulo o residuo).
* Relacionales: ==, !=, <, >, <=, >=
* Lógicos: && (AND), || (OR), ! (NOT)

Ejemplo:

int a = 10, b = 5;

if (a > b && b != 0) {

std::cout << "a es mayor y b no es cero";

}

Bloque 2 – Control de flujo

Condicionales  
Sirven para que el programa decida entre varias opciones.

if – else if – else

if (edad >= 18) {

std::cout << "Mayor de edad";

} else {

std::cout << "Menor de edad";

}

switch-case

int opcion = 2;

switch(opcion) {

case 1: std::cout << "Opción 1"; break;

case 2: std::cout << "Opción 2"; break;

default: std::cout << "Opción inválida";

}

Bucles  
Permiten repetir instrucciones varias veces.

for: cuando conocemos el número de repeticiones.

for (int i = 1; i <= 5; i++) {

std::cout << i << " ";

}

while: se repite mientras la condición sea verdadera.

int x = 0;

while (x < 3) {

std::cout << x << " ";

x++;

}

do-while: igual que while, pero se ejecuta al menos una vez.

int y = 0;

do {

std::cout << y << " ";

y++;

} while (y < 3);

Bloque 3 – Funciones

Qué son

* Una función es un bloque de código que realiza una tarea específica.
* Se define una sola vez y puede llamarse varias veces.
* Ayuda a organizar y reutilizar el código.

Tipos

1. Con retorno: devuelven un valor.

int suma(int a, int b) {

return a + b;

}

Se llama así:

int resultado = suma(4, 6);

1. Sin retorno (void): solo ejecutan instrucciones, no devuelven nada.

void saludar() {

std::cout << "Hola!";

}

Parámetros

* Son los valores que recibe la función para trabajar.
* Pueden ser de cualquier tipo: int, float, string, etc.

Bloque 4 – Arreglos y estructuras

Arreglos

* Un arreglo es una colección de variables del mismo tipo guardadas en posiciones consecutivas de memoria.
* Se accede a cada posición con un índice que comienza en 0.

Ejemplo:

int notas[3] = {7, 8, 9};

std::cout << "Primera nota: " << notas[0];

Estructuras (struct)

* Un struct agrupa diferentes variables (de distintos tipos) bajo un mismo nombre.
* Muy útil para representar objetos de la vida real (ej. un alumno con nombre y edad).

Ejemplo:

struct Alumno {

std::string nombre;

int edad;

};

Alumno a1;

a1.nombre = "Luis";

a1.edad = 19;

Aquí tienes más información más profunda sobre **Gitmoji** y ejemplos/preferencias de uso, junto con todo lo anterior formateado como texto plano para que lo copies en Word.

**Información adicional sobre Gitmoji: estándar, ejemplos, ubicación del emoji**

* Gitmoji es un proyecto (por Carlos Cuesta) que propone un conjunto estandarizado de emojis para prefixar mensajes de *commit*, de forma que al ver rápidamente el historial se entienda qué tipo de cambio hizo cada commit.
* Usos comunes: los emojis se colocan al **inicio** del mensaje del commit, antes del prefijo tipo o combinados (emoji + tipo + scope + descripción). Esto ayuda a que al hacer git log --oneline o revisar commits se identifique fácilmente el propósito.
* Otra convención usada (y recomendable si la quieres para tu clase) es combinar Gitmoji + Convencional Commit. Es decir:  
  <emoji> <tipo>(<scope>): <mensaje breve>  
  Ejemplo: :sparkles: feat(UI): Add new login button ([Gist](https://gist.github.com/alpteo/e93d754e5e09907c6362c4230fb66f87?utm_source=chatgpt.com" \o "Semantic Commit Messages with Emojis))
* Algunos emojis comunes y con qué tipo de prefijo combinan bien:

| **Tipo de commit (prefijo)** | **Emoji sugerido** | **Uso típico** |
| --- | --- | --- |
| feat (nueva funcionalidad) | ✨ (:sparkles:) | Al añadir una nueva función, característica, modulo nuevo. ([gitmoji.dev](https://gitmoji.dev/?utm_source=chatgpt.com" \o "gitmoji | An emoji guide for your commit messages)) |
| fix (arreglo de bug) | 🐛 (:bug:) | Cuando corriges un error. ([gitmoji.dev](https://gitmoji.dev/?utm_source=chatgpt.com" \o "gitmoji | An emoji guide for your commit messages)) |
| docs (documentación) | 📝 (:memo:) | Si actualizas README, ayudas, comentarios, guías. ([gitmoji.dev](https://gitmoji.dev/?utm_source=chatgpt.com" \o "gitmoji | An emoji guide for your commit messages)) |
| refactor | ♻️ (:recycle:) | Al reorganizar código sin cambiar funcionalidad. ([gitmoji.dev](https://gitmoji.dev/?utm_source=chatgpt.com" \o "gitmoji | An emoji guide for your commit messages)) |
| perf (rendimiento) | ⚡ (:zap:) | Cambios para optimizar velocidad, uso de recursos, etc. ([gitmoji.dev](https://gitmoji.dev/?utm_source=chatgpt.com" \o "gitmoji | An emoji guide for your commit messages)) |
| style (formato, estilo) | 🎨 (:art:) | Cuando ajustas estilo, formato, nombres, indentación, etc. ([gitmoji.dev](https://gitmoji.dev/?utm_source=chatgpt.com" \o "gitmoji | An emoji guide for your commit messages)) |
| test | ✅ (:white\_check\_mark:) | Al agregar o actualizar pruebas. ([gitmoji.dev](https://gitmoji.dev/?utm_source=chatgpt.com" \o "gitmoji | An emoji guide for your commit messages)) |
| build / ci | 🔧 (:wrench:) o 🏗️ (:construction\_worker:) | Cambios en scripts de build, configuración de CI/CD, dependencias. ([gitmoji.dev](https://gitmoji.dev/?utm_source=chatgpt.com" \o "gitmoji | An emoji guide for your commit messages)) |

* Sobre el orden: lo más usado es poner primero el emoji, luego el tipo, luego scope (opcional), luego dos puntos, luego mensaje breve. Ejemplo: ✨ feat(Auth): Add login feature
* También hay quienes lo ponen después del tipo, pero eso parece menos común; lo importante es consistencia dentro del equipo/proyecto.

Normas estilo Google C++ / buenas prácticas de nombrado, archivos, variables, funciones, estructuras

Nombre de archivos

* Usar nombres que reflejen el contenido.
* Archivos de cabecera (.h) y de implementación (.cc o .cpp) según el proyecto.
* Para headers, usar *include guards* cuyo nombre se base en la ruta dentro del proyecto, todo en mayúsculas con guiones bajos, para asegurar unicidad.

Variables, funciones, estructuras, tipos

* Los tipos como clases o structs usan CamelCase o PascalCase (cada palabra empieza con mayúscula).
* Funciones y variables empiezan con letra minúscula, usan lowerCamelCase.
* Constantes globales o macros usan mayúsculas con guiones bajos (SCREAMING\_SNAKE\_CASE).
* Enumeraciones, typedefs pueden variar según guía, pero muchas usan lower\_snake\_case o CamelCase dependiendo del estilo adoptado.

Constantes

* Constantes que no cambian tienen nombres en mayúsculas con guiones bajos.
* Usar constexpr cuando aplique.

Funciones

* Deben tener nombres que indiquen lo que hacen.
* Verbos si realizan acción.
* LowerCamelCase para nombre de funciones.

Reglas de commits en GitHub con prefijos

* Usar prefijos tipo: feat, fix, docs, refactor, test, ci, style, chore, perf.
* El prefijo se escribe seguido de :, y opcionalmente con un **scope** entre paréntesis después del prefijo para indicar sección o módulo afectado. Ejemplo: feat(auth): … o fix(parser): …
* Mensaje descriptivo breve y claro después del prefijo y posible scope.
* Primera línea del commit breve (< 50-72 caracteres), luego cuerpo si hay que explicar más.
* No combinar muchos tipos de cambios en un solo commit.

Gitmoji: estándar sugerido y ejemplos

* Gitmoji es guía que propone emojis estándar para prefijar commits según el tipo de cambio.
* Lo normal es poner el emoji al inicio del mensaje de commit. Ejemplo de formato: <emoji> <tipo>(<scope>): <mensaje>
* Ejemplos de emojis con tipos de commit:

✨ feat: nueva funcionalidad  
🐛 fix: corrección de bug  
📝 docs: documentación  
♻️ refactor: reorganización del código  
⚡ perf: mejoras de rendimiento  
🎨 style: cambios de estilo / formato  
✅ test: añadir o actualizar pruebas  
🔧 build / ci: cambios en build o integración continua / dependencias

* Ejemplos completos:

✨ feat(UI): Add login button  
🐛 fix(network): Handle timeout error  
📝 docs: Update README with usage  
♻️ refactor(utils): Simplify helper functions  
⚡ perf: Optimize image rendering  
✅ test: Add test for user input validation  
🔧 build: Update CI config file