Sistema de tipos

- El concepto de tipo es muy importante en C++
- Cada variable, argumento de función y valor devuelto por una función debe tener un tipo para compilarse
- Antes de evaluar cada una de las expresiones (incluidos los valores literales), el compilador da implícitamente un tipo a estas expresiones
- Algunos ejemplos de tipos son int, que almacena valores enteros, double, que almacena valores de punto flotante (también conocidos como tipos de datos escalares) o la clase std::basic_string de la biblioteca estándar, que almacena texto.

Sistema de tipos

- Puede crear su propio tipo definiendo un objeto **class** o **struct**
- El tipo especifica:
  - La cantidad de memoria que se asignará para la variable (o el resultado de la expresión)
  - Las clases de valores que se pueden almacenar en esa variable, cómo se interpretan estos valores (como patrones de bits) y las operaciones que se pueden realizar en ella.

Terminología

- **Variable**: nombre simbólico de una cantidad de datos.
  - Este nombre se puede utilizar para acceder a los datos a los que hace referencia en el ámbito del código en el que se define. En C++, el término “variable” se utiliza normalmente para hacer referencia a las instancias de tipos de datos.

- **Objeto**: por simplicidad y coherencia, el término “objeto” se usa para hacer referencia a cualquier instancia de una clase o estructura.

Especificar tipos de variable y función

- C++ es un lenguaje fuertemente tipado y que, además, contiene tipos estáticos. Cada objeto tiene un tipo y ese tipo nunca cambia (no debe confundirse con los objetos de datos estáticos)
- Al declarar una variable en el código, debe especificar explícitamente su tipo o utilizar la palabra clave auto para indicar al compilador que deduzca el tipo desde el inicializador
- Al declarar una función en el código, debe especificar el **tipo de cada argumento** y su **valor devuelto** (o void, si la función no devuelve ningún valor)
- La excepción se produce cuando se utilizan plantillas de función, que están permitidas en los argumentos de tipos arbitrarios
- Una vez que se declara por primera vez una variable, no se puede cambiar su tipo. Sin embargo, el valor de la variable o el valor devuelto por una función se puede copiar en otra variable de distinto tipo
  - Este tipo de operaciones se denominan conversiones de tipo. Estas conversiones a veces resultan necesarias, aunque también pueden producir errores o pérdidas de datos

Tipos de Datos

- Existen dos tipos
  - **Tipos simples llamados también primitivos**
  - **Tipos estructurados**

Tipos Simples (integrales)

- Son atómicos
- Ejemplos:
  - Número enteros, dobles (punto flotante), string de la biblioteca string.h para texto

- A diferencia de algunos lenguajes, C++ no tiene un tipo base universal del que se deriven todos los demás tipos. La implementación del lenguaje C++ contiene muchos tipos fundamentales, también conocidos como tipos integrados. Esto incluye los tipos numéricos, como **int, double, long y bool**

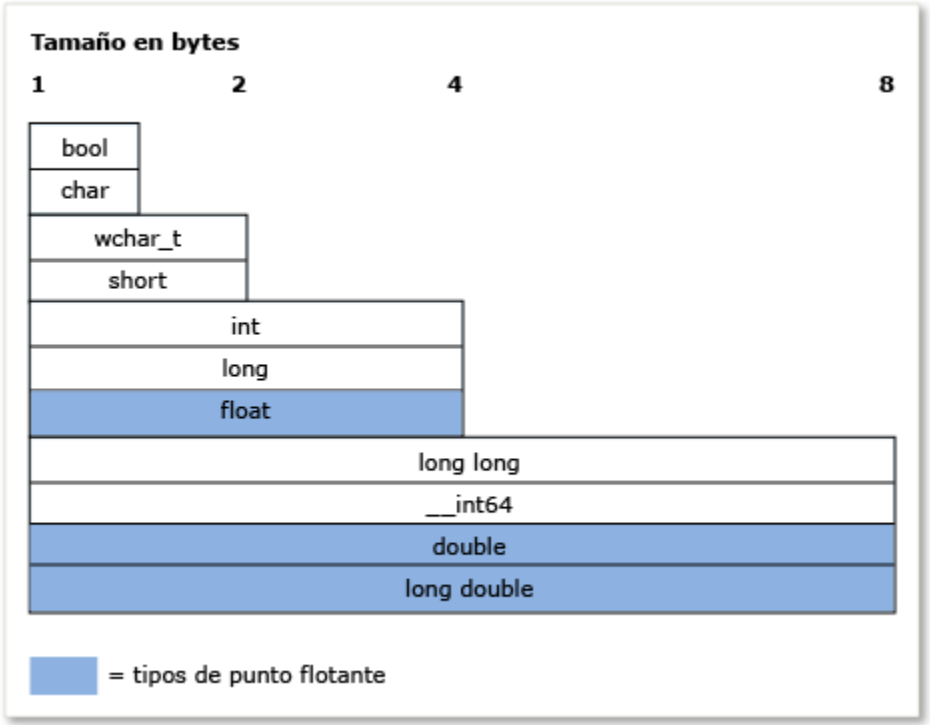
Tipos Simples (integrales)

La mayoría de los tipos fundamentales (excepto bool, double y tipos relacionados) tienen versiones **sin signo**, que modifican el intervalo de valores que la variable puede almacenar.

Por ejemplo, un valor int, que almacena un entero de **32 bits con signo**, puede representar un valor comprendido entre -2.147.483.648 y 2.147.483.647.

Un valor **unsigned int**, que también se almacena como **32 bits**, puede almacenar un valor comprendido entre 0 y 4.294.967.295. El número total de valores posibles en cada caso es el mismo; solo cambia el intervalo.

Tamaño relativo de los tipos



Tamaño relativo de los tipos

Tipo	Tamaño	Comentario
int	4 bytes	Opción predeterminada para los valores enteros.
double	8 bytes	Opción predeterminada para los valores de punto flotante.
bool	1 byte	Representa valores que pueden ser true o false.
char	1 byte	Se utiliza en los caracteres ASCII de cadenas de estilo C antiguas u objetos std::string que nunca tendrán que convertirse a UNICODE.
wchar_t	2 bytes	Representa valores de caracteres "anchos" que se pueden codificar en formato UNICODE (UTF-16 en Windows; puede diferir en otros sistemas operativos). Es el tipo de carácter que se utiliza en las cadenas de tipo `std::wstring`.
unsigned char	1 byte	C++ no tiene un tipo `byte` integrado. Utilice un carácter sin signo para representar un valor byte.
unsigned int	4 bytes	Opción predeterminada para los marcadores de bits.
long long	8 bytes	Representa valores enteros muy grandes.

Tamaño de los tipos enteros en rangos

- int (4 bytes): -2147483648 a 2147483647
- short int (2 bytes): -32768 a 32767
- long (long int): 8 bytes: -9223372036854775808 a 9223372036854775808
- unsigned int (4 bytes): 0 a 4294967295

Números enteros

- El más utilizado es el **int**
- Un entero largo es de tipo **long**

- Los siguientes modificadores permiten una definición más específica:
  - signed: indica que el valor máximo que se puede representar tiene por punto medio el 0 (por lo tanto se pueden representar valores negativos)
  - int es por defecto signed
  - unsigned: utilizado para poder representar números positivos
  - short: tipo con menos bytes
  - long: tipo que reserva más byte

Casting de tipos

- El casting es una operación en la cual se convierten valores de tipos compatibles
  - int a float
- Cuando se hace con tipos no compatibles ocurre un error de compilación
- Se utilizan los paréntesis y dentro de ellos el tipo de datos a convertir, por ejemplo:
  - int x = 10;
  - int y = 20;
  - float d1 = x / y; //El resultado es 0
  - float d2 = (float) x / y; //El resultados es 0.5

Tipos estructurados

- Los tipos estructurados son tipos que se componen de otros tipos
- Los tipos estructurados incluyen:
  - Arreglos (vectores)
  - String (cadenas)
  - Structs (estructuras o registros)

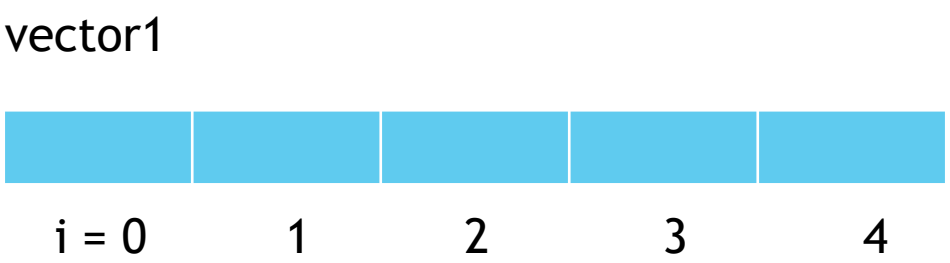
Arreglos

- Estructuras utilizadas para agrupar datos de forma estática en una única variable o propiedad
- Los arreglos permiten almacenar múltiples valores en posición de memoria continua, permitiendo acceder a la información de manera rápida y sencilla
- Se utilizan índices para el acceso a los valores
- Los valores de un arreglo dependerán de su declaración, pueden ser:
  - Números, booleanos, cadenas, Objetos

Arreglos

- Declaración de arreglos
- En C++ el arreglo inicia en la posición 0

  - int vector1[5];
  - float vector2[55];
  - string vector3[20];
  - bool vector4[10];
  - char vector5[4];



Arreglos

- Inicialización
- Especificando tamaño:
  - string cadenas[5] = {“saludos", “desde", “c", “++”}; //También inicializando
  - Persona *personas[5] = {NULL};
  - Persona *personas[5];
- Sin especificar tamaño:
  - int numeros[] = {10, 402, 30, 394, 394, 10, 55, 43};
- Accediendo al elemento en la posición 0
  - int x1 = numeros[0]; //Accediendo al valor 10
  - int x2 = numeros[100]; //Error en tiempo de ejecución

Tamaño de un arreglo

- En la declaración del arreglo se reserva memoria para las celdas continuas que almacenaras valores

- En C++ se utiliza la función sizeof para conocer los bytes utilizados por una variable
  - int nbytes = sizeof(personas);

  - int numeros[] = {39,43,14,33,11};
  - int nnumeros = sizeof(numeros); //20 bytes
  - int nenumeros = sizeof(numeros[0]); //4 bytes

Recorriendo un arreglo

- int numeros[] = {39,43,14,33,11};
- int tamanio = sizeof(numeros)/sizeof(numeros[0]);
- for (int i = 0; i < tamanio; i++){
  - cout << “[“ << i << “]” << Numeros[i];
- }

Actividad

- Prog 1. Hacer un programa que declare e inicialice un arreglo de enteros, y determine el número mayor del arreglo

- Prog 2. Hacer un programa que declare e inicialice un arreglo de enteros, y muestre los números que son primos.

- Prog3. Crear una clase Ferrocarril que administre un arreglo de tipo Vagón (clase) como propiedad. El arreglo en cada posición apuntará a un objeto tipo Vagón. El arreglo representará los vagones de los que se compone el ferrocarril. La clase Vagón deberá tener una propiedad donde indique la cantidad de pasajeros en el mismo, mientras que la clase ferrocarril deberá ofrecer los siguientes métodos:
  - Vagon* obtenerVagon(int indice); //Recuperar un vagón dada una posición
  - int obtenerNumeroDeVagones(); //Recuperar un vagón dada una posición
  - bool agregarVagon(Vagon *vagon); //Asignar en una posición libre del arreglo el vagón
    //enviado de parámetro