Алгоритмы, структуры данных и программирование

Типы данных и переменные в C++

Неделя 2 | Информационные системы, 2 курс

Lecture Objectives

1

Language Classification

Understand the classification of programming languages.

2

C++ Data Types

Know the basic data types in C++.

3

Memory Storage

Understand how data is stored in computer memory.

4

Data Type Selection

Be able to choose the appropriate data type for tasks.

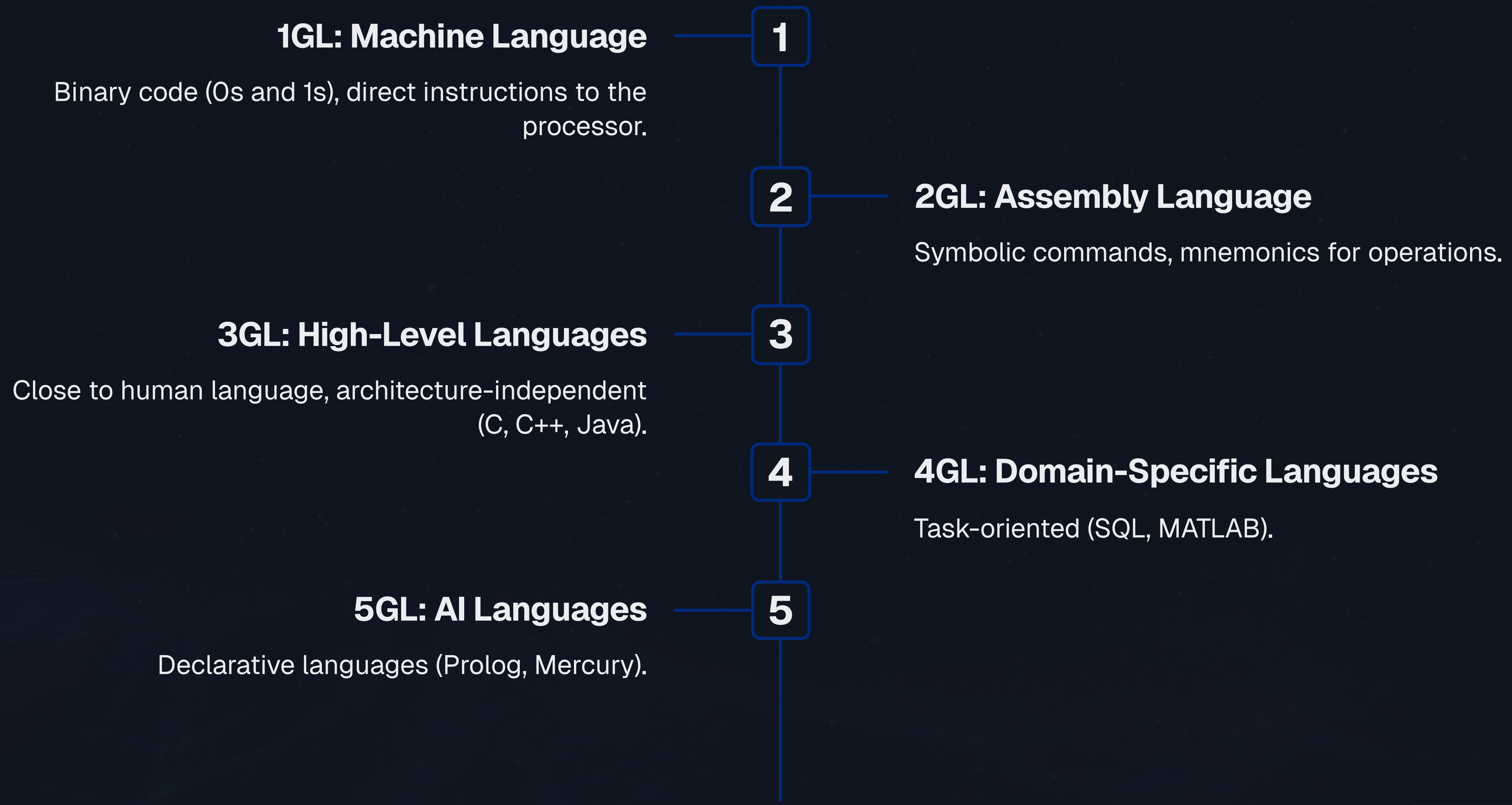
5

Number and Text Representation

Understand the principles of representing numbers and text in memory.

Programming Language Classification

A programming language is a formal language for writing computer programs, defining the rules of their appearance and the computer's actions.



1 **1GL: Machine Language**

Binary code (0s and 1s), direct instructions to the processor.

2 **2GL: Assembly Language**

Symbolic commands, mnemonics for operations.

3 **3GL: High-Level Languages**

Close to human language, architecture-independent (C, C++, Java).

4 **4GL: Domain-Specific Languages**

Task-oriented (SQL, MATLAB).

5 **5GL: AI Languages**

Declarative languages (Prolog, Mercury).

Compilation vs. Interpretation

Compiled

- Entire program translated
- Fast execution
- More complex to develop
- Examples: C, C++, Rust

Interpreted

- Line-by-line execution
- Slower execution
- Simpler to develop
- Examples: Python, JavaScript

C++ is a compiled language, which ensures high performance.

Data and Its Representation

Data is information presented in a formalized form suitable for computer processing. Computer memory is a sequence of numbered cells, each containing 1 byte of data.

Bit	1	Minimal unit (0 or 1)
Byte	8 bits	Basic unit of memory
Kilobyte (KB)	1024 bytes	~half a page of text
Megabyte (MB)	1024 KB	~minute of MP3 music
Gigabyte (GB)	1024 MB	~HD quality movie

Computer Memory

Computer memory consists of a sequence of numbered cells, each with a unique address. Each cell stores 1 byte of data, allowing it to hold a value from 0 to 255 (2^8 possible states).

Example Memory Cells

0x1000	42
0x1001	65
0x1002	0
0x1003	127
0x1004	255

Data Representation in Memory

Numbers

Numbers are stored in binary format (0s and 1s). For example, the decimal number 42 is represented as `00101010` in one byte.

Text

Text is represented as a sequence of characters, each encoded by a number (e.g., using ASCII or Unicode tables).

Images

Images consist of pixels, where each pixel has numerical values for its color components (e.g., RGB).

Sound

Sound is recorded by sampling, where the amplitude of the sound wave is measured at regular intervals and stored as numerical values.

Data Types in C++

In C++, every piece of data has a specific type, which plays a crucial role in how that data is stored and processed.

Set of Values

Defines the range of values a variable can hold (e.g., integers, characters, floating-point numbers).

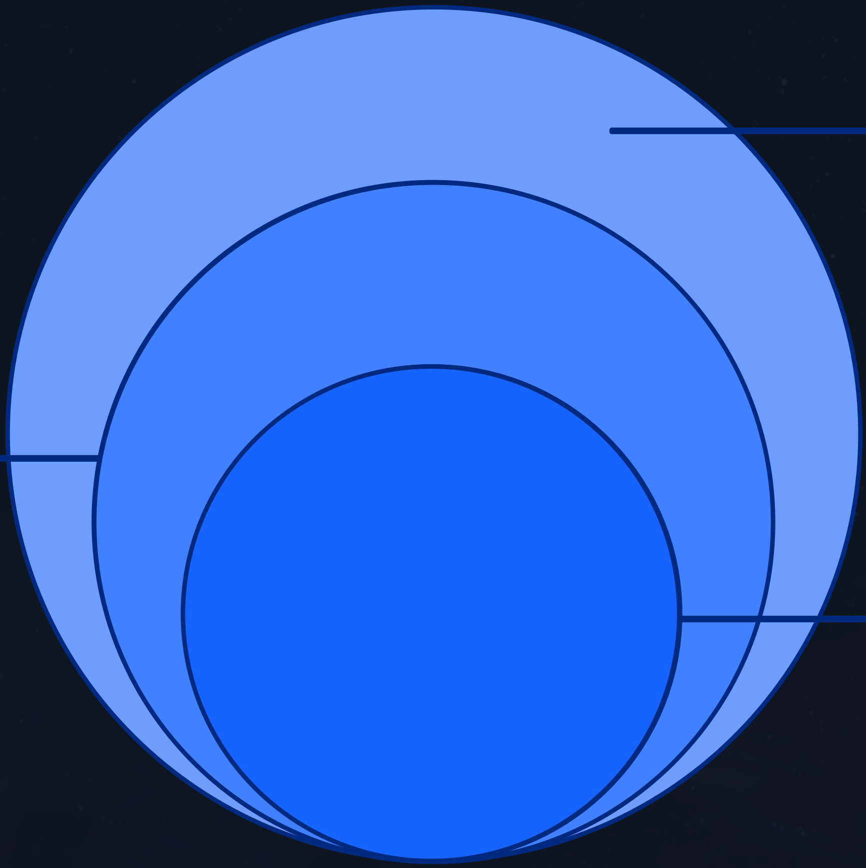
Memory Allocation

Specifies how much memory will be allocated to store the data (e.g., 1 byte for `char`, 4 bytes for `int`).

Set of Operations

Dictates which operations are applicable to the data (e.g., arithmetic operations for numbers, concatenation for strings).

Classification of Data Types



Derived

Pointers, References, Functions

Compound

Arrays, Structures, Unions, Classes

Fundamental

int, char, long, float, double, bool, void

Fundamental Data Types

Character	char	1 byte	-128 to 127
Character	unsigned char	1 byte	0 to 255
Integer	short	2 bytes	-32,768 to 32,767
Integer	int	4 bytes	-2,147,483,648 to 2,147,483,647
Integer	long long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Unsigned	unsigned short	2 bytes	0 to 65,535
Unsigned	unsigned int	4 bytes	0 to 4,294,967,295
Unsigned	unsigned long long	8 bytes	0 to 18,446,744,073,709,551,615
Floating-point	float	4 bytes	±3.4e-38 to ±3.4e+38
Floating-point	double	8 bytes	±1.7e-308 to ±1.7e+308
Floating-point	long double	10-16 bytes	±1.2e-4932 to ±1.2e+4932
Boolean	bool	1 byte	true/false

Overflow: Exceeding Type Limits

Overflow is a situation where the result of an arithmetic operation exceeds the range of values that a given data type can store. This can lead to unexpected and incorrect results.

```
unsigned char x = 255;  // Maximum for unsigned char
x = x + 1;              // Overflow!
// x is now 0 (wraps around)

signed char y = 127;    // Maximum for signed char
y = y + 1;              // Overflow!
// y is now -128
```

Unsigned Types

When an unsigned integer reaches its maximum value (e.g., 255 for `unsigned char`) and is incremented, it "wraps around" to its minimum value (0). This behavior is called **cyclic transition** or **wrap-around** and occurs due to the nature of binary arithmetic.

Signed Types

For signed integers, when the maximum positive value is reached (e.g., 127 for `signed char`) and incremented, it transitions to the minimum negative value (-128). This is called **signed integer overflow** and is related to the two's complement representation of numbers.

Floating-Point Types

Floating-point numbers are represented according to the IEEE 754 standard. They have limited precision.

float	32 bits	~7 digits
double	64 bits	~15 digits
long double	80-128 bits	~19 digits

Recommendations: use float for memory saving, double for precise calculations. Avoid equality comparisons, use epsilon.

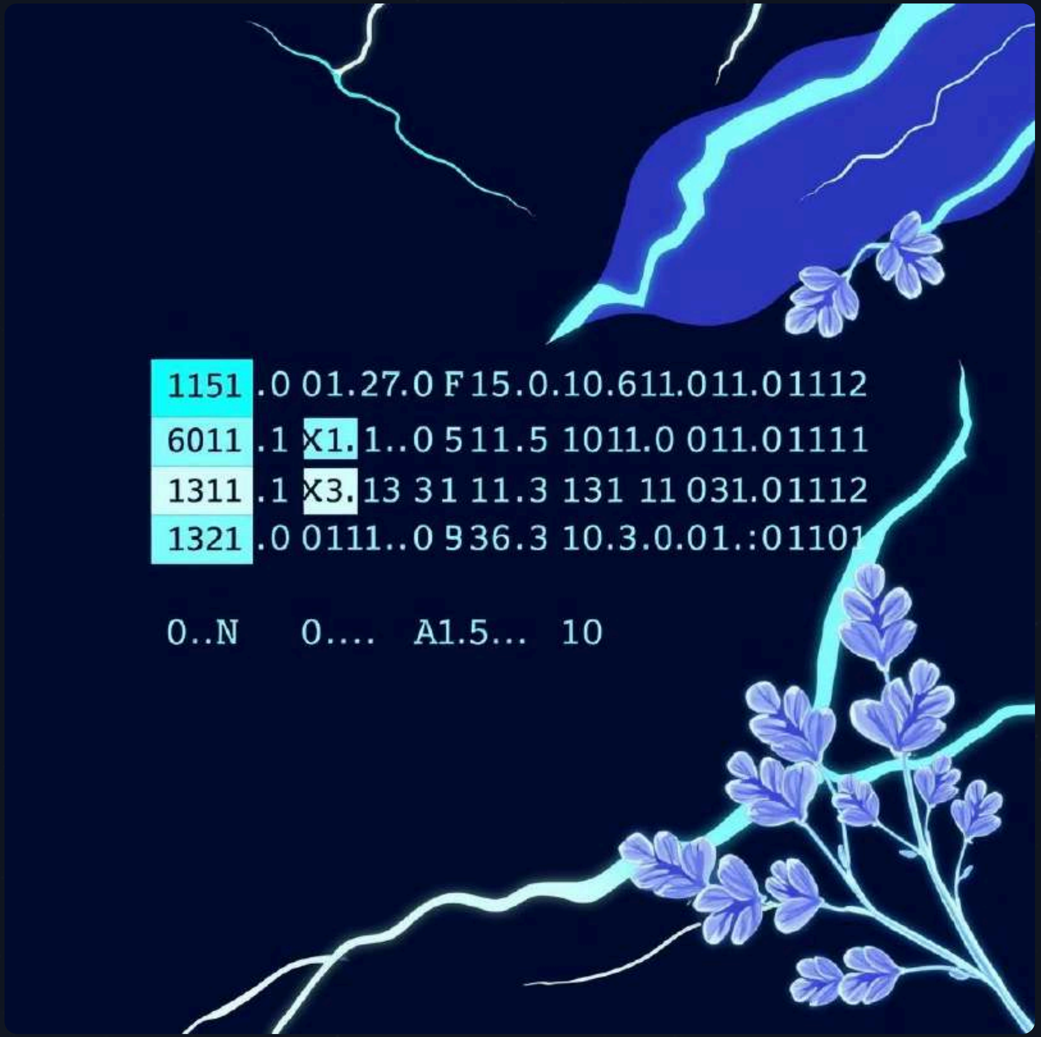
Character and Boolean Types

Char Type

- Size: 1 byte
- Stores character code (ASCII)
- Examples: 'A' (65), '0' (48), '\n' (10)

Bool Type

- Size: 1 byte
- Values: true (1) or false (0)
- Logical operations: && (AND), || (OR), ! (NOT)


1151 .0 01.27.0 F15.0.10.611.011.01112
6011 .1 X1. 1..0 511.5 1011.0 011.01111
1311 .1 X3. 13 31 11.3 131 11 031.01112
1321 .0 0111..0 536.3 10.3.0.01.:01101
0..N 0.... A1.5... 10

The correct choice of type is important for program efficiency and memory optimization.

Memory Optimization

Efficient memory usage is a key aspect when working with large amounts of data in C++. Choosing the correct data type for a variable can significantly reduce memory consumption and improve program performance.

```
// Suboptimal (16 bytes per record)
struct BadRecord {
    int day;        // 4 bytes (char would suffice)
    int month;      // 4 bytes (char would suffice)
    int year;       // 4 bytes (short would suffice)
    int count;      // 4 bytes
};

// Optimal (6 bytes per record)
struct GoodRecord {
    unsigned char day;     // 1 byte (1-31)
    unsigned char month;   // 1 byte (1-12)
    unsigned short year;   // 2 bytes (0-65535)
    unsigned short count;  // 2 bytes
};
```

Suboptimal Approach

Using `int` (4 bytes) for fields that can be represented by smaller types, such as `char` (1 byte) or `short` (2 bytes), leads to excessive memory consumption.

- `day`, `month`: could use `unsigned char`.
- `year`: could use `unsigned short`.

Optimal Approach

By choosing the minimally sufficient data types (e.g., `unsigned char` for day/month and `unsigned short` for year), we reduce the amount of memory for each record.

- `day` (1-31): `unsigned char` (1 byte).
- `month` (1-12): `unsigned char` (1 byte).
- `year` (0-65535): `unsigned short` (2 bytes).

In this example, optimization reduces the size of each record by 10 bytes. When working with a million records, this leads to significant savings of 10 MB, which is critical for high-performance systems.

Key Takeaways and Next Steps

1 **Language Classification**

By abstraction level and paradigm.

2 **C++**

Compiled, mid-level, OOP.

3 **Data Types**

Define size, range, and operations.

4 **Integers**

Negative numbers in two's complement.

5 **Floating-Point Numbers**

Limited precision.

6 **Characters**

Stored as numeric codes (ASCII).

7 **Type Selection**

Critical for efficiency.

Next Lecture: Control Structures (conditionals and loops).