
Lecture-3 — Fundamentals: Hardware → Data → C++ Types (Deep)

1) Introduction — pehli soch (first thought)

- Problem: Insaan ko quantities samajhni, store karni, aur calculate karni thi.
- Solution chain: counting → symbols → mechanical calculators → electronic computing → programming.
- Goal of this lecture: “Kaise physical world (transistor) se lekar C++ variable tak bridge banta hai.”

---

2) How transistor store data (simple, conceptual)

First thought: “Hume ek tiny switch chahiye jo ON/OFF state ko reliable tarike se rakhe” → transistor.

- **Transistor = electronic switch** (ON = 1, OFF = 0).
- Memory cell basics:
    - **SRAM (static)**: flip-flop (2 cross-coupled inverters) made from transistors — bit stable as long as power on.
    - **DRAM (dynamic)**: capacitor + transistor — capacitor holds charge → needs refresh.
- **Single bit** = one memory cell = transistor(s) state → 0/1.
- **Byte (8 bits)** = combination of 8 cells → store 0–255 or signed range with 2’s complement.

Blackboard sketch idea:

Bit cell (conceptual):
  [transistor switch] ----> 0 or 1
Flip-Flop (SRAM): stable latch with 2 inverters
DRAM: capacitor (charge/no charge) + transistor (access)

---

3) ASCII table (short, useful subset)

First thought: “Human letters ko machine ko kaise bataun?” → ASCII maps characters → numeric codes.

- ASCII standard: 7-bit (0–127). Common printable characters 32–126.
- Useful entries:
  - `65 = 'A'`, `66 = 'B'` … `90 = 'Z'`
  - `97 = 'a'`, `98 = 'b'` … `122 = 'z'`
  - `48 = '0'` … `57 = '9'`
  - `32 = ' '` (space), `10 = LF` (newline)
- C++: `char c = 'A'; int x = (int)c; // 65`

---

4) First code in C++ (why, then code)

First thought: “Give the machine a simple instruction and show result on screen.”
Classic first program:

```
#include <iostream>
int main() {
    std::cout << "Hello, World!\n";
    return 0;
}
```

- `#include <iostream>` → bring in I/O library (cout).
- `main()` → program entry point.
- `std::cout` prints text to console.

---

5) Code on computer screen — how it becomes running program

Pipeline (conceptual):

1. **Write source** (`.cpp`) in editor.
2. **Compile**: compiler (`g++`) → preprocess → compile → assemble → link → produces executable.
3. **Run**: OS loads executable → CPU executes instructions.

Flow: `source.cpp` → (compiler) → `a.out` → run.

---

6) Variable and data type — first thought

- **Problem:** We need named storage (human-readable) for values in memory.
- **Variable = name → memory location**.
- **Data type = contract**: how many bytes, how to interpret bits, what ops allowed.

---

7) `int` (in C++)

- Typical: 4 bytes (32 bits) on common platforms (range −2,147,483,648 … 2,147,483,647).
- Use when whole numbers fit in that range.

Declaration:
int age = 20;

-

Memory view (example `age = 5`):
[byte3][byte2][byte1][byte0] => binary of 5

-

---

8) `char`

- 1 byte (in C++ usually 1 byte), stores character code (ASCII), or small ints −128..127 (signed char) or 0..255 (unsigned char).

Example:
char ch = 'A'; // stores 65

-
- Printing: `std::cout << ch;` or `std::cout << int(ch);` to see numeric code.

---

9) `float`

- 4 bytes (IEEE-754): approx 6–7 decimal digits precision. Good for decimals but **not** money.

Example:
float f = 3.14159f;

-
- Memory: sign(1) + exponent(8) + mantissa(23).

---

10) `double`

8 bytes (IEEE-754): approx 15–16 decimal digits precision. Use when more precision needed.
double d = 3.141592653589793;

-

---

11) `bool`

- Logical type: `true` / `false`.
- Stored typically as 1 byte but conceptually one bit.

Example:
bool isEven = true;

-

---

12) How to use variable and data in code (step by step)

First thought: Name the storage, choose type, initialize, use.

Example — sum two numbers:

```
#include <iostream>
using namespace std;
int main() {
    int a = 10;
    int b = 20;
    int sum = a + b;
```

```
    cout << sum; // 30
    return 0;
}
```

Dry-run: CPU reads values from memory cells of `a` and `b`, ALU does addition, result stored in `sum`, printed.

---

13) Negative / Positive integer storage (signed integers)

First thought: Need to represent + and − with same bits.

- Signed integers use **2's complement** (common).
- For N bits: range = −2^(N−1) .. 2^(N−1) − 1.
  Example 8-bit: −128 .. +127.

Example: represent −5 in 8-bit 2's complement:

- +5 = `00000101`
- 1's complement = `11111010`
- 2's complement = add 1 → `11111011` => −5

---

14) 1's and 2's complement explained (step)

- **1's complement**: flip all bits. Problem: two zeros (`0000` and `1111`), arithmetic awkward.
- **2's complement**: 1's complement + 1 → single zero, plus simple addition/subtraction hardware.

Visual (8-bit):

```
+5  = 00000101
1's complement = 11111010
2's complement = 11111011  // -5
```

Why needed: makes subtraction same as addition with negative numbers → simpler ALU.

Lecture-4 — Input, Operators, Casting, Conditionals, Loops (Hands-on C++)

---

1) How to take input from user & how to use `cin`? (first thought)

- **First thought:** machines need data at runtime—give a keyboard interface → program reads from standard input stream.
- In C++: `cin` (istream) + extraction operator `>>`.

Example:

```
#include <iostream>
using namespace std;
int main() {
    int x;
    cout << "Enter x: ";
    cin >> x; // waits for user, reads from keyboard buffer
    cout << "You entered " << x << '\n';
    return 0;
}
```

Working: user types text + Enter → OS buffer → `cin` extracts token, converts to chosen type, stores in variable.

---

2) Take user input to sum two numbers (hands-on)

Why: interactive programs.

```
#include <iostream>
using namespace std;
int main() {
    int a, b;
    cout << "Enter two integers: ";
    cin >> a >> b;
    cout << "Sum = " << (a + b) << '\n';
```

```
    return 0;
}
```

Dry run: user enters `5  7` → `a=5`, `b=7` → prints `12`.

---

3) Operators in C++

Categories & examples (first thought: what ops do we need?)

- **Arithmetic:** `+ - * / %`
- **Assignment:** `=, +=, -=, *=, /=, %=`
- **Relational:** `==, !=, >, <, >=, <=`
- **Logical:** `&&, ||, !`
- **Bitwise:** `&, |, ^, ~, <<, >>`
- **Increment/Decrement:** `++, --`
- **Ternary:** `?:`

---

4) Typecasting concept (review + deep)

First thought: sometimes you must change the interpretation of bits from one type to another.

- **Implicit (promotion):** safe widening (e.g., `int -> double`) done by compiler.
- **Explicit (cast):** programmer forces conversion: `(type)value` or `static_cast<type>(value)`.

Examples:

```
double d = 3;          // implicit: int -> double
int n = (int)3.99;     // explicit: truncates -> 3
int m = static_cast<int>(3.99);
```

Prefer `static_cast` in C++ for clarity.

---

5) Data loss concept (detailed)

When/why data lost:

- **Narrowing conversions:** float → int (fraction lost), double → float (precision lost), long → int (overflow if out of range).
- **Overflow:** value outside target type range wraps (undefined/implementation-defined for signed types in C/C++ older standards; but in practice for unsigned it wraps modulo).
- **Precision:** floating point stores finite precision — big integers can't be exactly represented.

Example:

```
double d = 123456789012345.0;
float f = static_cast<float>(d); // precision loss
```

---

6) Type casting hands-on (example)

```
#include <iostream>
using namespace std;
int main() {
    double d = 9.99;
    int i = (int)d; // 9
    cout << i << '\n';
    long big = 3000000000LL;
    int small = static_cast<int>(big); // likely overflow
    cout << small << '\n';
}
```

Discuss: show expected outputs & reason (overflow behavior platform-dependent for signed ints).

---

7) If-else condition start (first thought)

First thought: branch program flow based on condition (decision diamond in flowchart).

Syntax:

```
if (condition) {
  // true branch
} else {
  // false branch
```

}

---

8) If-else example with salary package

Problem: If salary >= 50000 → no increment else give 10% increment.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
    double salary;
    cout << "Enter salary: ";
    cin >> salary;
    if (salary >= 50000) {
        cout << "No increment\n";
    } else {
        salary *= 1.10;
        cout << fixed << setprecision(2);
        cout << "New salary: " << salary << '\n';
    }
    return 0;
}
```

Dry run: input `45000` → output `49500.00`.

---

9) Marks grading system (logic)

- Suppose: 90–100 → A, 75–89 → B, 60–74 → C, 40–59 → D, <40 → F.

---

10) Hands-on marks grading system (code)

```
#include <iostream>
using namespace std;
int main() {
    int marks;
    cout << "Enter marks (0-100): ";
```

```
    cin >> marks;
    if (marks >= 90) cout << "Grade A\n";
    else if (marks >= 75) cout << "Grade B\n";
    else if (marks >= 60) cout << "Grade C\n";
    else if (marks >= 40) cout << "Grade D\n";
    else cout << "Fail\n";
    return 0;
}
```

Dry run: input `78` → Grade B.

---

11) If-else rules (good practices)

- Conditions evaluate left→right as written; use parentheses for clarity.
- Avoid floating equality checks (`==`) for double; use epsilon.
- Use `else if` to check ranges; last `else` for fallback.
- Keep branch bodies small — call functions for complex tasks.

---

12) Compare two variables (code)

```
int a, b;
cin >> a >> b;
if (a > b) cout << "a is greater\n";
else if (a < b) cout << "b is greater\n";
else cout << "equal\n";
```

---

13) Hands-on (compare exercise) — dry run

Input: `5 7` → prints `b is greater`.

---

14) Check number even or odd

Logic: `n % 2 == 0` → even, else odd.

```
int n; cin >> n;
if (n % 2 == 0) cout << "Even\n"; else cout << "Odd\n";
```

---

15) Homework: voter eligibility

Problem statement: If age >= 18 → eligible else not.

```
int age; cin >> age;
if (age >= 18) cout << "Eligible\n"; else cout << "Not eligible\n";
```

---

16) Number positive, negative, or zero

```
int n; cin >> n;
if (n > 0) cout << "Positive\n";
else if (n < 0) cout << "Negative\n";
else cout << "Zero\n";
```

---

17) Hands-on (positive/negative) — dry run

Input `0` → prints `Zero`.

---

18) Character is vowel or not

Check: lower/upper vowels `'a','e','i','o','u'` and caps.

```
char c; cin >> c;
c = tolower(c);
if (c=='a' || c=='e' || c=='i' || c=='o' || c=='u') cout << "Vowel\n";
else cout << "Consonant\n";
```

---

19) Print weekday with number

Mapping 1..7 → Mon..Sun. Use `switch` (clean).

```
int d; cin >> d;
switch(d) {
  case 1: cout<<"Monday\n"; break;
  case 2: cout<<"Tuesday\n"; break;
  case 3: cout<<"Wednesday\n"; break;
  case 4: cout<<"Thursday\n"; break;
  case 5: cout<<"Friday\n"; break;
  case 6: cout<<"Saturday\n"; break;
  case 7: cout<<"Sunday\n"; break;
  default: cout<<"Invalid\n";
}
```

---

20) Loop concept introduction (first thought)

- **Problem:** Repeat work many times — avoid code duplication.
- Loops let program **iterate**: `for`, `while`, `do-while`.

---

21) Syntax of `for` loop

```
for (initialization; condition; update) {
  // body
}
```

---

22) Explain the working of `for` loop (step)

1. Initialization executed once.
2. Condition checked — if false, loop ends.
3. Body executed.
4. Update executed.
5. Go to step 2.

Example trace: `for (int i=1;i<=3;i++)` → i=1 body, i=2 body, i=3 body, i=4 condition false -> exit.

---

23) Print number 1 to 5

```
for (int i=1; i<=5; ++i) cout << i << ' ';
```

Output: `1 2 3 4 5`

---

24) Print square 1 to n

```
int n; cin >> n;
for (int i=1; i<=n; ++i) cout << (i*i) << ' ';
```

---

25) Print "coder army" 10 times

```
for (int i=0;i<10;i++) cout << "coder army\n";
```

---

26) Print n natural numbers

```
int n; cin >> n;
for (int i=1;i<=n;i++) cout << i << ' ';
```

---

27) Hands-on: print squares & all even numbers up to 20

- Squares 1..20: loop i=1..20 print i*i.
- Even numbers up to 20: `for(i=2;i<=20;i+=2)` print i.

---

28) Code: print even numbers up to 20

```
for (int i=2;i<=20;i+=2) cout << i << ' ';
```

Output: `2 4 6 8 10 12 14 16 18 20`
