MTH 4300/4299: Programming and Computer Science II

Lecture 02: C++ Basics

Jaime Abbariao

Data types

Unlike Python, C++ requires that we either have types that are easily inferable or explicit.

For example, in Python, we could initialize variables as the following:

```python
name = "jaime"
another_name: str = "jaime"
```

However, in C++, if we wanted to do something similar, we'd have to do the following:

```cpp
auto name = "jaime"; // the type for `name` is inferred from the right-side
std::string another_name = "jaime";
```

int

`int` represents the integer data type:

Type	Size	Range
`int`	At least 16 bits (typically 32)	-2,147,483,648 to 2,147,483,647
`unsigned int`	At least 16 bits (typically 32)	0 to 4,294,967,295
`short` / `short int`	At least 16 bits	-32,768 to 32,767

If a value exceeds these limits, you'll encounter overflow (undefined behavior).

```
int a;                // declared, but uninitialized
int b = 1;            // initialized to 1
auto c = 1;           // type inferred as int

int d, e, f;          // multiple declarations
int g = 1, h, i = 1;  // partial initialization
```

Note: Uninitialized variables contain garbage values. Always initialize your variables.

long, long int, long long, and long long int

When you need to store larger integers, C++ provides extended integer types:

Type	Minimum Range
long / long int	At least 32 bits
long long / long long int	At least 64 bits

```
long population = 8000000000L;        // L suffix for long literals
long long distance_to_sun = 149597870700LL; // LL suffix for long long
```

Note: long and long int are equivalent, as are long long and long long int.

float and double

Floating-point types store decimal numbers:

Type	Precision	Size
float	~7 decimal digits	4 bytes
double	~15 decimal digits	8 bytes

```
float pi_approx = 3.14159f;    // f suffix for float literals
double pi = 3.1415926535897793; // default for decimal literals

double result = 10.0 / 3.0;    // 3.333...
```

Important: Prefer double over float unless memory is a concern. Floating-point arithmetic can introduce rounding errors.

bool

A `bool` stores a boolean value: either `true` or `false`. It typically occupies 1 byte:

```
bool is_valid = true;
bool has_error = false;
auto is_ready = true;  // type inferred as bool
```

Booleans are commonly used with comparison and logical operators:

```
int x = 5, y = 10;

bool result1 = (x < y);    // true
bool result2 = (x == y);   // false
bool result3 = (x != y);   // true

bool both = (x > 0) && (y > 0);   // logical AND: true
bool either = (x > 10) || (y > 5); // logical OR: true
bool not_valid = !is_valid;        // logical NOT: false
```

Note: In C++, non-zero integers convert to `true`, and zero converts to `false`.

char

A `char` stores a single character and occupies 1 byte. Characters are enclosed in single quotes:

```
char letter = 'A';
char digit = '7';
char newline = '\n';  // escape sequence
```

Under the hood, `char` stores an integer (ASCII value):

```
char ch = 'A';
int ascii_value = ch;  // ascii_value is 65

char next = 'A' + 1;   // next is 'B'
```

Common escape sequences: `\n` (newline), `\t` (tab), `\\` (backslash), `\'` (single quote)

▒▒ std::string

`std::string` is C++'s standard string class. Unlike `char`, strings use double quotes:

```
#include <string>

std::string greeting = "Hello, World!";
std::string empty;           // empty string ""
std::string repeated(5, 'x'); // "xxxxx"
```

Useful operations:

```
std::string s = "Hello";
s.length();        // 5
s.size();          // 5 (same as length)
s[0];              // 'H'
s + " World";      // "Hello World" (concatenation)
s.substr(1, 3);    // "ell" (start index, length)
```

Type Casting

Implicit conversion

This happens automatically when the compiler converts one type to another without explicit instruction

```
double d = 3; // int 3 is implicitly converted to double
```

static_cast

This requires explicit syntax to convert between types.

Only allows conversions that are checked and considered safe at compile time.

```
int i = static_cast<int>(3.14); // double 3.14 explicitly converted to int
```

Operators

Arithmetic Operators

Operator	Description	Example
+	Addition	5 + 3 → 8
-	Subtraction	5 - 3 → 2
*	Multiplication	5 * 3 → 15
/	Division	5 / 3 → 1 (integer), 5.0 / 3.0 → 1.666...
%	Modulus (remainder)	5 % 3 → 2

```
int a = 10, b = 3;
int sum = a + b;        // 13
int quotient = a / b;   // 3 (integer division truncates)
int remainder = a % b;  // 1

double x = 10.0, y = 3.0;
double result = x / y;  // 3.333...
```

Note: Integer division truncates toward zero. The modulus operator % only works with integers.

Operators (ii)

Comparison Operators

These return a `bool` value (`true` or `false`):

Operator	Description	Example
`==`	Equal to	`5 == 5` → `true`
`!=`	Not equal to	`5 != 3` → `true`
`<`	Less than	`3 < 5` → `true`
`>`	Greater than	`5 > 3` → `true`
`<=`	Less than or equal	`5 <= 5` → `true`
`>=`	Greater than or equal	`5 >= 3` → `true`

```
int x = 5, y = 10;
bool is_equal = (x == y);    // false
bool is_less = (x < y);      // true
bool is_greater = (x >= y);  // false
```

Operators (iii)

Assignment Operators

Operator	Equivalent To	Example
=	Assignment	x = 5
+=	x = x + y	x += 3
-=	x = x - y	x -= 3
*=	x = x * y	x *= 3
/=	x = x / y	x /= 3
%=	x = x % y	x %= 3

```
int x = 10;
x += 5;   // x is now 15
x *= 2;   // x is now 30
x %= 7;   // x is now 2
```

Operators (iv)

Increment and Decrement Operators

Operator	Description	Example
++x	Pre-increment: increment, then return	++x
x++	Post-increment: return, then increment	x++
--x	Pre-decrement: decrement, then return	--x
x--	Post-decrement: return, then decrement	x--

```
int a = 5;
int b = ++a;  // a is 6, b is 6 (increment first, then assign)
int c = a++;  // a is 7, c is 6 (assign first, then increment)

int x = 10;
std::cout << x++ << std::endl;  // prints 10, x is now 11
std::cout << ++x << std::endl;  // prints 12, x is now 12
```

Logical Operators

Operator	Description	Example
&&	Logical AND	true && false → false
||	Logical OR	true || false → true
!	Logical NOT	!true → false

```
bool a = true, b = false;

bool and_result = a && b;   // false (both must be true)
bool or_result = a || b;    // true (at least one must be true)
bool not_result = !a;       // false

// Short-circuit evaluation
int x = 0;
if (x != 0 && 10 / x > 1) {  // second condition never evaluated
    // ...
}
```

Short-circuit evaluation: && stops if the first operand is false; || stops if the first operand is true.

▒▒ Operator Precedence

Operators are evaluated in a specific order. Higher precedence operators are evaluated first:

Precedence	Operators
Highest	`++ --` (postfix)
	`++ --` (prefix), `!`, unary `+ -`
	`* / %`
	`+ -`
	`< <= > >=`
	`== !=`
	`&&`
	`||`
Lowest	`= += -= *= /= %=`

```
int result = 2 + 3 * 4;     // 14, not 20 (multiplication first)
int clarified = 2 + (3 * 4); // 14 (explicit precedence)
```

When in doubt, use parentheses to make your intentions clear.

▒▒ Input and Output

C++ uses streams for input and output. The <iostream> header provides:

Stream	Purpose
std::cout	Standard output (console)
std::cin	Standard input (keyboard)
std::cerr	Standard error output

```
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

The << operator is called the insertion operator (sends data to the stream).

▒▒ Output with std::cout

Use << to chain multiple outputs:

```
int age = 25;
std::string name = "Alice";

std::cout << "Name: " << name << std::endl;
std::cout << "Age: " << age << std::endl;

// Chaining on a single line
std::cout << name << " is " << age << " years old." << std::endl;
```

Output:

```
Name: Alice
Age: 25
Alice is 25 years old.
```

std::endl outputs a newline and flushes the buffer. You can also use '\n' for just a newline.

▒▒ Input with std::cin

Use >> (the extraction operator) to read input:

```
#include <iostream>

int main() {
    int age;
    std::cout << "Enter your age: ";
    std::cin >> age;
    std::cout << "You are " << age << " years old." << std::endl;
    return 0;
}
```

std::cin automatically converts input to the target type and skips leading whitespace.

```
int a, b;
std::cin >> a >> b;  // reads two integers separated by whitespace
```

▒▒ Reading Strings

`std::cin >> variable` reads until whitespace. For full lines, use `std::getline`:

```cpp
#include <iostream>
#include <string>

int main() {
    std::string word, line;

    std::cout << "Enter a word: ";
    std::cin >> word;  // reads until whitespace

    std::cin.ignore();  // clear the newline left in buffer

    std::cout << "Enter a full line: ";
    std::getline(std::cin, line);  // reads entire line

    std::cout << "Word: " << word << std::endl;
    std::cout << "Line: " << line << std::endl;
    return 0;
}
```

Note: After using `std::cin >>`, call `std::cin.ignore()` before `std::getline` to clear the leftover newline.

▒▒ Input Validation

`std::cin` returns a reference that converts to `false` if input fails:

```
#include <iostream>

int main() {
    int number;
    std::cout << "Enter an integer: ";

    if (std::cin >> number) {
        std::cout << "You entered: " << number << std::endl;
    } else {
        std::cout << "Invalid input!" << std::endl;
        std::cin.clear();  // clear error state
        std::cin.ignore(10000, '\n');  // discard bad input
    }
    return 0;
}
```

Always validate user input in real programs to handle unexpected data gracefully.

▒▒ Lab Exercise: Temperature Converter

Create a program that converts temperatures between Fahrenheit and Celsius.

Requirements:

1. Prompt the user for a temperature value (as a `double`)
2. Ask the user which conversion they want:
   - `1` for Fahrenheit to Celsius
   - `2` for Celsius to Fahrenheit
3. Perform the conversion using the appropriate formula:
   - Celsius = (Fahrenheit - 32) × 5/9
   - Fahrenheit = Celsius × 9/5 + 32
4. Display the result with a descriptive message

Starter code:

```
#include <iostream>

int main() {
    double temperature;
    int choice;

    // TODO: Prompt user for temperature

    // TODO: Prompt user for conversion choice (1 or 2)

    // TODO: Perform conversion based on choice

    // TODO: Display the result

    return 0;
}
```

Lab Exercise: Expected Output

```
Enter a temperature: 100
Choose conversion (1: F to C, 2: C to F): 1
100 F = 37.7778 C
```

```
Enter a temperature: 0
Choose conversion (1: F to C, 2: C to F): 2
0 C = 32 F
```

Bonus challenges:

1. Add input validation: display an error if the user enters an invalid choice
2. Use `static_cast` to demonstrate converting the result to an `int` (truncated value)

Submission

You will be allowed to submit your solutions to this problem on Brightspace until tomorrow at noon