１回目　導入回＋文字列

第1回授業：C++導入とCとの違い（入出力・文字列・書式制御）

☒ 目標

C言語の知識をもとに、C++の基本文法と文字列処理を理解する。

- `iostream` を使った標準入出力を理解する。
- `std::string` と `char[]` の違いを学ぶ。
- `cout` による書式制御を体験する。

☒ 授業構成（90分想定）

時間	内容	ポイント
0〜10分	CとC++の関係	「C++はCの拡張」だが、文字列処理や入出力が高機能であることを紹介。
10〜30分	入出力の基本	`#include <iostream>`、`std::cout` / `std::cin` の使い方。`using namespace std;` の意味。
30〜55分	文字列の扱い	`char[]` と `std::string` の違いを説明。代入・連結・比較などを実演。
55〜80分	書式制御	`std::setw`、`std::setfill`、`std::fixed`、`std::setprecision` など `<iomanip>` の使用。表形式出力の練習。
80〜90分	まとめ・課題説明	文字列と入出力のまとめ。次回予告：「関数と参照」へ。

☒ 授業のねらい

- C言語で扱いづらかった「文字列」や「入出力フォーマット」がC++で簡潔に扱えることを体感させる。
- 今後の授業で使う `std::string` を習得し、OOPでのクラス利用にもスムーズに入れるようにする。

☒ 板書・スライド例

Cの例（文字列と出力）

```
#include <stdio.h>
int main() {
    char name[32];
    printf("Input your name: ");
    scanf("%s", name);
    printf("Hello, %s!\n", name);
    return 0;
}
```

C++の例（std::string版）

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    string name;
    cout << "Input your name: ";
    cin >> name;
    cout << "Hello, " << name << "!" << endl;
    return 0;
}
```

☒ ポイント

- `string` は動的にサイズが変わる安全な文字列クラス。
- 代入・連結・比較が簡単。

```cpp
string a = "Hello";
string b = "World";
string c = a + ", " + b;
if (a == "Hello") cout << c << endl;
```

☒ 比較:

操作	C (char[])	C++ (std::string)
代入	`strcpy(s, "abc")`	`s = "abc";`
結合	`strcat(a, b)`	`a += b;`
比較	`strcmp(a, b) == 0`	`a == b`
長さ	`strlen(a)`	`a.size()`

✏ 書式制御の例

```cpp
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    double pi = 3.14159265;
    cout << fixed << setprecision(3);
    cout << "pi = " << pi << endl;

    cout << setw(10) << setfill('-') << 42 << endl;
    return 0;
}
```

☒ ポイント

- `<iomanip>` によりCの `printf` に近い整形出力が可能。
- `setw()`：桁幅指定、`setfill()`：埋め文字、`setprecision()`：小数点桁数指定。

☒ 演習課題

1. 名前と年齢を入力し、整形して出力するプログラムを作成。
   ```
   Name: _____
   Age : __ years old
   ```
2. `std::string` を使って2つの単語を入力し、結合して表示する。
3. `<iomanip>` を使って3つの数値を整列表示する。

☒ 授業後課題（宿題）

1. `std::string` と `char[]` の違いを、3つの観点（代入・結合・安全性）で説明せよ。
2. 書式制御を使って表のようなスコア表示を作るプログラムを書け。

☒ 次回予告

> **第2回：関数と参照・オーバーロード**
> C++独自の関数設計（参照渡し・デフォルト引数）を学び、関数の柔軟性を理解する。

補足資料

第1回授業補足編：書式制御とfmtライブラリ入門

☒ 目的

C++における出力整形の幅を広げ、より実践的なフォーマット出力を理解する。

- `<iomanip>` による標準的な書式指定を整理。
- `std::format`（C++20）や `fmt` ライブラリの書式構文を紹介。
- `printf` との対応関係を示し、Cからの移行をスムーズにする。

☒ <iomanip>による標準フォーマットまとめ

操作	操作関数	効果	使用例

桁幅指定	`setw(n)`	指定幅に右寄せ	`cout << setw(5) << 42;`
			→
			`42`

左寄せ	`left`	出力を左寄せ	`cout << left << setw(5) << 42;`
			→
			`42`

埋め文字	`setfill(c)`	空白を指定文字で埋める	`cout << setfill('-') << setw(5) << 42;`
			→
			`---42`

進数指定	`dec`
	/
	`hex`
	/
	`oct`
		10進・16進・8進表示	`cout << hex << 255;`
			→
			`ff`

小数点桁数	`setprecision(n)`	小数点以下の桁数指定	`cout << fixed << setprecision(2) << 3.14159;`
			→
			`3.14`

浮動小数点表記	`fixed`
	/
	`scientific`
		固定小数点／指数表記	`cout << scientific << 0.00123;`
			→
			`1.23e-03`

☒ ポイント

- `setw()` は **次の出力にのみ有効**（毎回指定が必要）。
- `setfill()` は **継続して有効**。
- `fixed` を指定しない場合、整数部分や小数点以下が自動的に最適化される。

☒ 応用例：表形式出力

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    cout << left << setw(10) << "Name" << right << setw(8) << "Score" << endl;
```

```
    cout << setfill('-') << setw(18) << "" << endl;
    cout << setfill(' ');

    cout << left << setw(10) << "Alice" << right << setw(8) << 95 << endl;
    cout << left << setw(10) << "Bob"   << right << setw(8) << 87 << endl;
    cout << left << setw(10) << "Chris" << right << setw(8) << 100 << endl;
}
```

出力例：

```
Name       Score
------------------
Alice         95
Bob           87
Chris        100
```

☒ std::format (C++20) とfmtライブラリ

C++20以降では、Python風の書式指定が可能な`std::format`が導入されています。旧環境では`fmt`ライブラリ（https://github.com/fmtlib/fmt）を使用できます。

使い方例

```
#include <format> // C++20以降
#include <iostream>
using namespace std;

int main() {
    string name = "Alice";
    int score = 95;
    double rate = 0.8765;

    cout << format("Name: {:<10} | Score: {:>4d} | Rate: {:.2f}\n", name, score, rate);
}
```

出力：

```
Name: Alice      | Score:   95 | Rate: 0.88
```

書式指定子	意味	例
`:<10`	左寄せ・幅10	`{: <10}` → 左寄せ
`:>10`	右寄せ・幅10	`{: >10}` → 右寄せ
`:.2f`	小数点以下2桁	`{:.2f}` → `3.14`
`:06d`	6桁ゼロ埋め	`{:06d}` → `000042`
`{}`	自動推論	デフォルト出力

☒ fmtライブラリでの使用例（C++17以前）

```
#include <fmt/core.h>
#include <string>

int main() {
    std::string name = "Bob";
    int age = 20;
    fmt::print("Hello, {}! You are {} years old.\n", name, age);
}
```

出力：

```
Hello, Bob! You are 20 years old.
```

☒ 比較：Cのprintf関数との違い

操作	printf	iostream	fmt / format
書式構文	`%d %f %s`	関数チェーン形式	`{}` ベースのテンプレート式
型安全性	☒ 弱い	☒ 強い	☒ 強い
可読性	△	○	◎
柔軟性	○	○	◎（動的組み立て可）

☒ まとめ

- `<iomanip>` は既存Cライクコードに馴染む標準方式。
- `std::format` / `fmt` は現代的で読みやすく、ゲームデバッグ出力などに最適。
- どちらも「**Cのprintfより安全で拡張性が高い**」という点を体験させるのが狙い。

☒ 次にやるとよい演習

1. 数値を右寄せ・左寄せしてスコア表を整形。
2. `fmt::print` を使ってデバッグメッセージを出すツール関数を作る。
3. `std::format` でログ出力を作り、`cout` 版と比較する。

Revision #2
★Created 8 October 2025 01:58:51 by youe2
Updated 9 June 2026 07:37:11 by youe2