# C++プログラミングI

- 第7回：ファイル操作
- 担当：二瓶芙巳雄

# 標準入出力とファイル

- 標準入出力ストリーム用の変数とデフォルト動作
  - `cin`：キーボードからの入力を扱う
  - `cout` 画面への出力を扱う
  - `cerr`：画面への出力を扱う（エラー出力）
- リダイレクション
  - プログラムの標準入出力をファイルにつなげる機能
  - プログラムを実行する際の指定
  - `cin`，`cout`，`cerr` の先が変わる
  - コマンド実行時に `<` や `>` の記号で指定する
- パイプ
  - プログラムの標準入出力を他プログラムにつなげる
  - プログラムを実行する際の指定
  - `cin`，`cout`，`cerr` の相手先が変わる
  - コマンド実行時に `|` の記号で指定する

# 標準入出力の接続先変更の例

- `cout` の出力先を `result.txt` に変える例
  - ※右図参照

```
$ ./a.out > result.txt
```

- `cerr` の出力先は画面のまま

- `cin` の入力元を `result.txt` に変える例

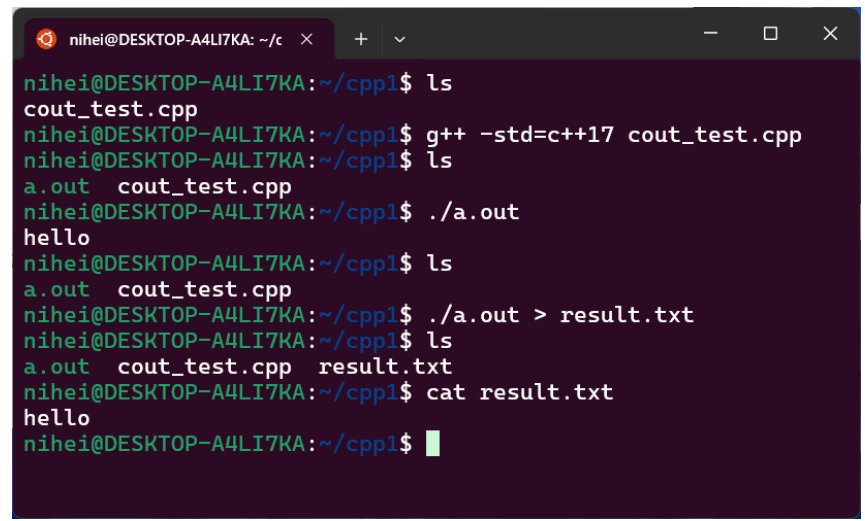
```
$ ./a.out < result.txt
```

- **不等号の向きに注意！**

- `./a.out` の標準出力の先を `wc` の標準入力に繋ぐ

```
$ ./a.out | wc
```

```cpp
// cout_test.cpp
#include <iostream>
int main() {
  std::cout << "hello\n";
}
```



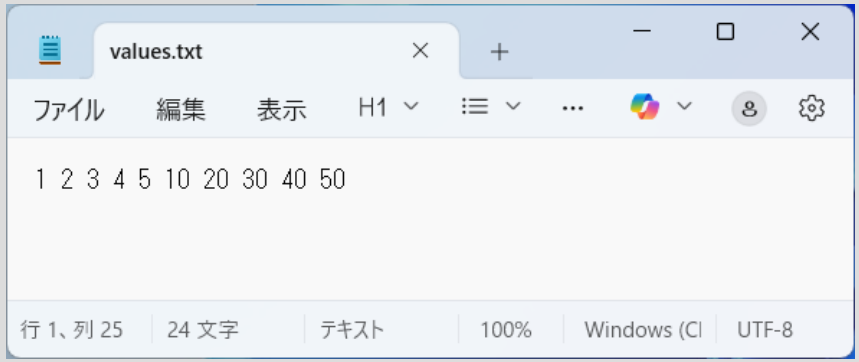

**プログラムの出力をテキストファイルに残せて便利！**

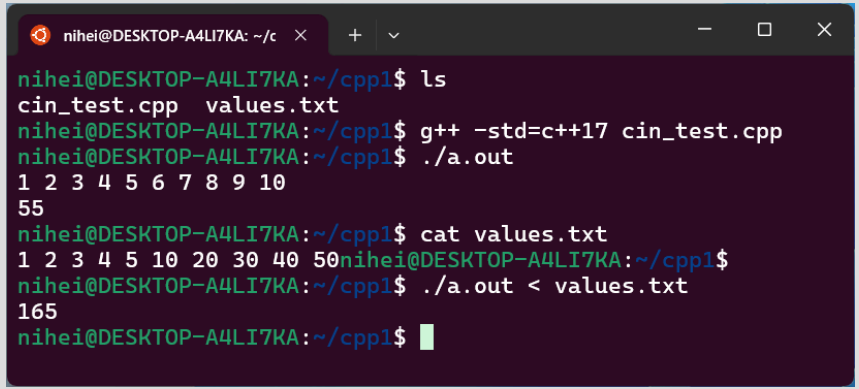
# ファイルからの入力リダイレクション：具体例

- `cin` の入力元を `result.txt` に変える例

```
$ ./a.out < result.txt
```

```
1  // cin_test.cpp
2  #include <iostream>
3  int main() {
4    int total = 0;
5    for( int i = 0; i < 10; i++ ) {
6      int x;
7      std::cin >> x;
8      total += x;
9    }
10   std::cout << total << "\n";
11 }
```




**プログラムへの手打ち入力がなくなって便利！**

# 指定ファイルからのデータ入力

- 複雑なことをする場合は，ファイル入出力のためのプログラムを書くほうが便利
- `<fstream>` ヘッダファイルを指定する
- 入力ファイル用に `ifstream` 型の変数を宣言する
- エラーチェックをする
- `cin` と同じ方法で入力を扱う

```cpp
#include <iostream>
#include <fstream>

int main() {
  std::ifstream fin {"file02.dat"};
  if (!fin) {
    std::cerr << "cannot open\n";
    return 1;
  }

  for (int x{}; fin >> x; ) std::cout << x <<" ";
}
```

```
$ ls
main.cpp
$ g++ -std=c++17 main.cpp -o f2.out
$ ls
main.cpp  f2.out
$ ./f2.out
cannot open
$ echo 1 2 3 4 5 6 7 8 > file02.dat # ファイル作成
$ ls
main.cpp  f2.out  file02.dat
$ cat file02.dat
1 2 3 4 5 6 7 8
$ ./f2.out
1 2 3 4 5 6 7 8
```

- 初回はファイルがないのでエラー出力
- `echo` 出力をリダイレクトしてファイル作成
  - シェルだけでファイルを作るテクニック
  - メモ帳でファイル作ってもOKです
- `cat` で中身を確認して実行

# ファイルの拡張子

- `a.out`？ `result.txt`？ `file02.dat`？
  - 末尾3文字なに？
- 拡張子：ファイル名の末尾 `.` 以降の（大体）3文字のやつ
  - プログラミングでよく見る: `.cpp` `.py` `.txt` `.h` `.dat`
- 拡張子は，OSがファイルを「開く」ときに，参考にする記号
  - OS「`.txt`とか`.cpp`とかならメモ帳で開いてあげるね」
  - OS「`.mp3`とか`.avi`とかならメディアプレイヤーで開いてあげるね」
- 拡張子はファイルの実態を表さない
  - `.cpp`にPythonのプログラムを書くことも可能
  - `.cpp`にはC++のプログラムを書く，と人が決めているだけ
- 拡張子を過信せず，とりあえずメモ帳（などのテキストエディタ）で開く癖をつけましょう

# 指定ファイルへのデータ出力

- `<fstream>` ヘッダファイルを指定する(入力と同じ)
- 出力ファイル用に `ofstream` 型の変数を宣言する
- エラーチェックをする（ファイルを作成できない領域にファイルを作ろうとするとエラー）
- `cout` と同じ方法で出力を扱う

```cpp
#include <iostream>
#include <fstream>

int main(int argc, char *argv[]) {
  std::ofstream fout {"output.dat"};
  if (!fout) {
    std::cerr <<"cannot open\n";
    return 1;
  }

  for (int i = 1; i <= 10; i++)
    fout << i <<": "<< i*i <<", ";
}
```

```
$ ls
main.cpp
$ g++ -std=c++17 main.cpp
$ ls
main.cpp  a.out
$ ./a.out
$
$ ls
main.cpp  a.out  output.dat
$ cat output.dat
1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100,
```

# ファイル入力と標準入出力

- 入力データをファイルから読み込み
- 検索データのをキーボードから読み込む
- プロンプト（ユーザ入力を促す文．以下の例では `search for:` ）と検索結果を画面に表示

```cpp
// ファイル内に任意の数値があるかどうかを確かめるプログラム
#include <fstream>
int main(int argc, char *argv[]) {
  std::ifstream fin {"file04.dat"};
  if (!fin) { /*エラー処理*/ }

  std::vector<int> data(100); int num {0};
  while (num < data.size() && fin >> data[num]) ++ num;

  for (int x{}; std::cout << "search for: " && std::cin >> x; ) {
    int i;
    for (i = 0; i < num; i++) if (data[i] == x) break;

    cout << x <<" is " << ((i < num) ? "" : "not ") << "found\n";
  }
}
```

```
$ cat file04.dat
1 2 3 4 5 6 7 8
$ ./a.out
search for: 3
3 is found
search for: 5
5 is found
search for: 100
100 is not found
search for: (Crtl+Dを入力)
$
```

# 多数データを読み込む場合

- `push_back()`の利用

### `push_back()`を利用したコード

```cpp
#include <fstream>
int main(int argc, char *argv[]) {
  std::ifstream fin {"file04.dat"};
  if (!fin) { /*エラー処理*/ }

  std::vector<int> data; int x {0};
  while ( fin >> x ) data.push_back(x);

  for (int x{};
       std::cout << "search for: " && std::cin >> x; ) {
    int i;
    for (i = 0; i < data.size(); i++) if (data[i] == x) break;

    cout << x <<" is "
         << ((i < data.size()) ? "" : "not ") << "found\n";
  }
}
```

### 修正前（1ページ前）のコード

```cpp
#include <fstream>
int main(int argc, char *argv[]) {
  std::ifstream fin {"file04.dat"};
  if (!fin) { /*エラー処理*/ }

  std::vector<int> data(100); int num {0};
  while (num < data.size() && fin >> data[num]) ++ num;

  for (int x{};
       std::cout << "search for: " && std::cin >> x; ) {
    int i;
    for (i = 0; i < num; i++) if (data[i] == x) break;

    cout << x <<" is "
         << ((i < num) ? "" : "not ") << "found\n";
  }
}
```

# 入力用ファイル名の読み込み

- 処理対象のファイルをキーボード入力で指定
- `ifstream` や `ofstream` に対して以下が可能
  - 文字列リテラルの指定
  - `string` 型の変数の指定

```cpp
#include <fstream>
int main()  {
  // ファイル名の入力
  std::string nm;
  std::cout << "input file name ---> ";
  std::cin >> nm;

  // 入力用ファイルオブジェクトの準備
  std::ifstream fin {nm};
  if (!fin) { /*エラー処理*/ }

  // データの読み込みと改行をつけた出力
  for (int data{}; fin >> data; )
    std::cout << data <<"\n";
}
```

```
$ ls
a.out  main.cpp  data1.txt  data2.txt  data3.txt
$ cat data1.txt
1 2 3 4 5
$ cat data2.txt
6 7 8 9 10
$ ./a.out
input file name ---> data1.txt
1
2
3
4
5
$ ./a.out
input file name ---> data2.txt
6
7
8
9
10
```

# 複数ファイルからのデータ入力

- ループの繰り返しごとに異なったファイルを扱う
- `in` 変数はループごとに別ファイルで初期化される．`in` 変数のスコープに注意．
- ※先ほどまでは変数名を `fin` にしてましたが，あくまでも変数名なので，自由に変更が可能です

```cpp
#include <fstream>
int main(int argc, char *argv[]) {
  std::vector<std::string> file_list {
    "data1.txt", "data2.txt", "data3.txt"
  };

  int total {0};
  for (auto file : file_list) {
    std::cout << "target: " << file << "\n";

    std::ifstream in {file};
    if (!in) { return 0; }

    for (int data{}; in >> data; ) total += data;
  }
  std::cout <<"total = "<< total <<"\n";
}
```

```
$ ls
a.out  data1.txt  data2.txt  data3.txt  main.cpp
$ cat data1.txt
1 2 3 4 5
$ cat data2.txt
6 7 8 9 10
$ cat data3.txt
11 12 13 14 15
$ ./a.out
target: data1.txt
target: data2.txt
target: data3.txt
total = 120
```

# コマンド引数の利用

- 半開区間 `argv+1, argv+argc` の指定
- `argv` の場所は実行ファイル名の文字列

```
1   int main(int argc, char* argv[]) {
2     vector<string> file_list(argv+1, argv+argc);
3
4     std::cout << "target: ";
5     for( std::string file: file_list )
6       std::cout << file << " ";
7     std::cout << "\n";
8
9     int total {0};
10    for (auto file : file_list) {
11      std::ifstream in {file};
12      if (!in) { /*エラー処理*/ }
13      for (int data{}; in >> data; )
14      total += data;
15    }
16    std::cout <<"total = "<< total <<"\n";
17  }
```

※コマンド引数は「第5回講義」を参照

```
$ ls
a.out  data1.txt  data2.txt  data3.txt  main.cpp
$ cat data1.txt
1 2 3 4 5
$ cat data2.txt
6 7 8 9 10
$ cat data3.txt
11 12 13 14 15
$ ./a.out data1.txt data2.txt data3.txt
target: data1.txt data2.txt data3.txt
total = 120
```

- コマンド引数の中で `?` を指定すると，その部分の1文字だけが異なるファイルを探し，コマンド引数として展開

```
$ ls
a.out  data1.txt  data2.txt  data3.txt  main.cpp
$ cat data?.txt
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
$ ./a.out data?.txt
target: data1.txt data2.txt data3.txt
total = 120
```

# 余談：`main()`の引数とvector

```
1  int main(int argc, char* argv[]) {
2    vector<string> file_list(argv+1, argv+argc);
3  }
```

- `argc`：コマンドライン引数の数．コマンド名も引数に含まれるため，argc は「引数の数 + 1」
  - 例）`./a.out data1.txt data2.txt data3.txt data4.txt` では，`argc == 5`
- `argv`：コマンドライン引数として与えられた文字列
  - `argv[0] == ./a.out`，`argv[1] == data1.txt`，`argv[2] == data2.txt`，以下略
- `std::vector<std::string>(start, end)`：c配列の `start` から `end` の範囲内の要素で vector を作成
  - `argv + 1`：ポインタ `argv` の先頭を1つ進めた位置（つまり `argv[1]` ）
  - `argv + argc`：ポインタ `argv` の終端（つまり `argv[5]` ）
    - 配列の範囲外を指すが，C++ では範囲指定では許容される
  - `argv[1]` - `argv[5]` の要素を使用して，`file_list` を初期化することになる
- この例では最終的に，`file_list[0] == "data1.txt"`，`file_list[1] == "data2.txt"`，`file_list[2] == "data3.txt"`，`file_list[3] == "data4.txt"` になる

# ファイルオブジェクトの引数

- リファレンス指定のみ．`const`は付けない．

```
1  #include <fstream>
2
3  void print( std::ofstream &o, int i)  { // 第一引数が ofstream, constなしで，&を付ける
4    for (int j = 0; j < 20; j++) o << i*j << "\n";
5  }
6
7  int main(int argc, char *argv[]) {
8    string filename {"output.txt"};
9    std::ofstream out {filename};
10   if (!out) { return 0; }
11
12   print( out, 5 ); // 実引数としてofstreamのoutを渡す
13 }
```

# `std::cout` にも対応する関数

- `std::ofstream` ではなく，`std::ostream` を型名に使う
- `ofstream` 変数は `ostream` に指定可能

修正後

```
1  void print(std::ostream &o, int i) {
2    for (int j = 0; j < 20; j++)
3      o << (i+1)*j << "\n";
4  }
5
6  int main() {
7    string filename {"output.txt"};
8    std::ofstream out {filename};
9    if (!out) { /*エラー処理*/ }
10
11   print(std::cout, i); // 画面に出力（表示）
12   print(out, i);       // ファイルに出力
13 }
```

修正前

```
1  void print(std::ofstream &o, int i) {
2    for (int j = 0; j < 20; j++)
3      o << (i+1)*j << "\n";
4  }
5
6  int main() {
7    string filename {"output.txt"};
8    std::ofstream out {filename};
9    if (!out) { /*エラー処理*/ }
10
11   // print(std::cout, i); // エラー！
12   print(out, i);          // ファイルに出力
13 }
```

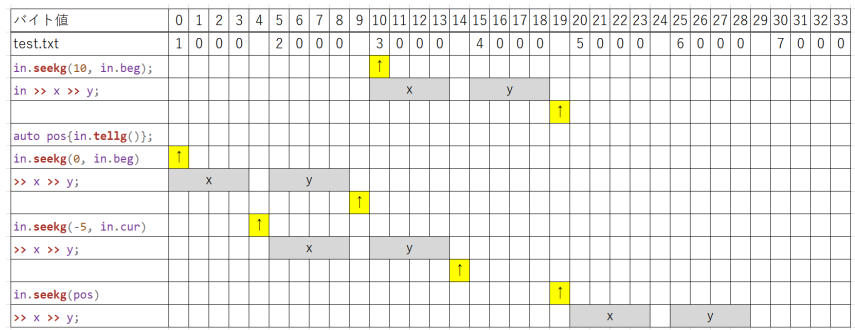
# 読み込み位置の変更

- ファイルの特徴
  - ファイルは先頭から読まなくても良い
  - 一度読んだ場所に戻って再度読み込んでも良い
- メンバ関数の利用
  - `.seekg(オフセット, 起点)`：次の入力を起点からバイト単位のオフセット分だけずれた場所から読み込むように設定する。起点は以下のどれかを指定する
    - `.beg`：ストリームの先頭 (begin)
    - `.cur`：現在位置 (current)
    - `.end`：ストリームの末尾 (end)
    - ※上記3つは数値ではない
  - `.seekg(先頭からの位置)`：`tellg()`の結果を使い読み込む場所を直接指定
    - ※第二引数を省略した場合は，起点が`beg`で固定されるイメージ
  - `.tellg()`：現在の読み込み位置の取得（整数型）

# 読み込み位置変更の例

```cpp
int main(int argc, char *argv[]) {
  std::ifstream in {"test.txt"};
  if (!in) { return 0; }

  int x, y;
  in.seekg(10, in.beg); // 先頭から 10 バイト
  in >> x >> y;
  std::cout << "A: " << x <<" "<< y <<"\n";

  auto pos { in.tellg() }; // 読み込み位置を保存

  in.seekg(0, in.beg) >> x >> y; // 先頭
  std::cout << "B: " << x <<" "<< y <<"\n";

  in.seekg(-5, in.cur) >> x >> y; // 5 バイト戻る
  std::cout << "C: " << x <<" "<< y <<"\n";

  in.seekg(pos) >> x >> y; // pos の位置
  std::cout << "D: " << x <<" "<< y <<"\n";
}
```

```
$ ls
a.out  main.cpp  test.txt
$ cat test.txt
1000 2000 3000 4000 5000 6000 7000
$ ./a.out
A: 3000 4000
B: 1000 2000
C: 2000 3000
D: 5000 6000
```




- ASCIIコードの範囲（アルファベットや記号，数など）は全て1バイト
- 日本語の文字 (SHIFT-JIS, UTF-8) はマルチバイト

# 読み込み位置変更の例（図示）

- 見づらいときはこちらをご覧ください

| バイト値 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| test.txt | 1 | 0 | 0 | 0 | | 2 | 0 | 0 | 0 | | 3 | 0 | 0 | 0 | | 4 | 0 | 0 | 0 | | 5 | 0 | 0 | 0 | | 6 | 0 | 0 | 0 | | 7 | 0 | 0 | 0 |
| `in.seekg(10, in.beg);` | | | | | | | | | | | ↑ | | | | | | | | | | | | | | | | | | | | | | | |
| `in >> x >> y;` | | | | | | | | | | | x | | | | | y | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | ↑ | | | | | | | | | | | | | |
| `auto pos{in.tellg()};` | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| `in.seekg(0, in.beg)` | ↑ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| `>> x >> y;` | x | | | | | y | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | ↑ | | | | | | | | | | | | | | | | | | | | | | | | |
| `in.seekg(-5, in.cur)` | | | | | ↑ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| `>> x >> y;` | | | | | | x | | | | | y | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | ↑ | | | | | | | | | | | | | | | | | | | |
| `in.seekg(pos)` | | | | | | | | | | | | | | | | | | | | ↑ | | | | | | | | | | | | | | |
| `>> x >> y;` | | | | | | | | | | | | | | | | | | | | | x | | | | | y | | | | | | | | |

# ファイルサイズの取得

- ファイルの末尾に移動
- 次に読み込む位置はファイルサイズと同じ

```
1  int main(int argc, char *argv[]) {
2    std::string file { "test.txt" };
3    std::ifstream in { file };
4    if (!in) { return 0; }
5
6    in.seekg(0, in.end);
7    auto size { in.tellg() };
8    std::cout << file <<" "<< size <<"\n";
9  }
```

```
./a.out
test.txt 35
```

# 書き込み位置の変更

- `.tellp()`：現在の書き込み位置の取得
- `out.end`：ストリームの末尾
- 書き込むバイト数を確保する必要がある
  - `std::setw()` で幅を指定

```
1   // データを読み込んだ個数countをファイルの先頭行に書く
2   #include <fstream>
3   #include <iomanip>
4
5   int main(int argc, char *argv[]) {
6     std::ofstream out {"out.txt"};
7     if (!out) { return 0; }
8
9     out << "data file\n";
10    auto pos { out.tellp() };
11
12    for (int count{1}, x; std::cin >> x; count++) {
13      out.seekp(pos) << std::setw(5) << count << "\n";
14      out.seekp(0, out.end) << x <<" ";
15    }
16    out <<"\n";
17  }
```

```
$ ./a.out
0 2 4 6 8 <<-- 入力後に Ctrl-D
$ cat out.txt
data file
    5
0 2 4 6 8
```

1回目のループ

```
data file
    1
0
```

2回目のループ

```
data file
    2
0 2
```

3回目のループ

```
data file
    3
0 2 4
```

...以下省略

# 追加書き込み

- 変数宣言時の指定で追加書き込みが可能
- ファイル名の他に `std::ios::app` を指定する

```
1  std::ofstream out{"outlog.txt",std::ios::app};
2  if (!out) {
3     std::cout << "cannot open\n";
4     return 1;
5  }
6  out << "append data\n";
```

- 指定は次のどれでも良い
  - `std::ios_base::app`
  - `std::ios::app`
  - `std::ostream::app`
  - `std::ofstream::app`

# （参考）リダイレクションによる追記

```
$ rm result
$ echo 1 2 3 >> result
$ echo 4 5 6 >> result
$ echo 7 8 9 >> result
$ cat result
1 2 3
4 5 6
7 8 9
```

# pythonとの比較：リダイレクション

```cpp
// c++
int main() {
  int a, b;
  std::cin >> a >> b;

  std::cout << "1.5x: " << a*1.5 << " " << b*1.5 << "\n";
}
```

```
$ g++ -std=c++17 main.cpp
$ ./a.out
3 5
1.5x: 4.5 7.5
$
$ cat values.txt
11 13
$ ./a.out < values.txt
1.5x: 16.5 19.5
$
$ ./a.out > result.txt
7 9
$ cat result.txt
1.5x: 10.5 13.5
```

```python
# python
line = input()
a = int( line.split()[0] )
b = int( line.split()[1] )

print("1.5x:", a * 1.5, b * 1.5)
#
```

```
$
$ python main.py
3 5
1.5x: 4.5 7.5
$
$ cat values.txt
11 13
$ python main.py < values.txt
1.5x: 16.5 19.5
$
$ python main.py > result.txt
7 9
$ cat result.txt
1.5x: 10.5 13.5
```

# pythonとの比較：ファイル入出力

```cpp
// c++
#include <iostream>
#include <fstream>

int main() {
  std::ifstream fin{"input.txt"};   // 入力ファイルを開く
  std::ofstream fout{"output.txt"}; // 出力ファイルを開く

  if (!fin) return 1;   // エラー処理
  if (!fout) return 1;  // エラー処理

  int a, b;
  fin >> a >> b;

  fout << "1.5x: " << a * 1.5 << " " << b * 1.5 << "\n";
}
```

```python
# python



try:
  fin = open("input.txt", "r")
  fout = open("output.txt", "w")

except FileNotFoundError: # エラー処理
  return 1

line = fin.read()
a, b = int( line.split()[0] ), int( line.split()[1] )

fout.write(f"1.5x: {a * 1.5} {b * 1.5}\n")
#
```