C++プログラミングI

- 第3回：string とvector
- 担当：二瓶芙巳雄

複数の値を扱うデータ型

- 配列：複数の値をまとめて格納するデータ構造
- C言語の配列：
  - C++でも使用できる
  - 最低限の管理機能しか提供されていない
  - 使用するには様々な知識必要
    - ポインタとメモリ管理
- C++の配列：
  - **string**型: 文字列を扱うためのデータ型
  - **vector**型: C配列を拡張した汎用の配列

string型

`string` 型

- `string` 型: 文字列を扱うための型．基本データ型ではなく，標準ライブラリに定義されたクラス
  - クラス: 自作できる型（のようなもの）
- `<string>` ヘッダファイルをインクルード
- ※ C言語で文字列を扱うには，`char` 型の配列 (C-string) を使っていた
- 文字列リテラルはC-string
  - ヌル文字で終端する配列 (p.8)
  - 文字が並ぶ範囲をどう表すかが重要

```
1  #include <iostream>
2  #include <string> // iostreamがあるなら省略可
3  int main() {
4    std::string s {"Hello!"}; // 文字列リテラル
5    std::cout << s <<"\n";    // cout の利用
6  }
```

```
% ./a.out
Hello!
```

`string`: 初期化，代入，入力

- `string` の初期化や代入に文字列リテラルを使用する
- 文字列リテラルは `string` 型ではない
  - 初期化・代入時に，型変換されている

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

int main() {
  std::string s  {"Hello!"}; // 文字列リテラル
  std::string s2 {s};        // 他変数の初期値
  std::string s3;            // ""（空文字列）での初期化

  s = "How are you?";        // 文字列リテラルの代入
  s3 = s2;                   // 代入，C言語配列では不可

  std::cin >> s2;            // cin も利用できる
}
```

`string`: 連結

- `+` 演算子は文字列の連結
- `+=` 演算子で更新も可能
- 以下に注意（`s` は `string` 変数とする）
  - 不可：`"abc" + "xyz"`（リテラルのみ連結）
    - `s += "abc" + "xyz"` も同様に不可
  - 可能：`s + "abc" + "xyz"`, `"abc" + s + "xyz"`
  - これはどうなるか？：`"abc" + "xyz" + s`

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

int main() {
  std::string s2 {"Hello"};
  std::string s3 {};

  s3 = s2 + "! ";       // 文字列リテラル
  s3 = s3 + s3;         // 変数
  std::cout << s3 <<"\n";

  s2 += ", thank you,"; // 文字列の追加
  s3 = " and You";
  s2 += s3;             // 変数の追加
  s2 += '?';            // 文字の追加
  std::cout << s2 <<"\n";
}
```

```
% ./a.out
Hello! Hello!
Hello, thank you, and You?
```

`string` 型変数の比較

- 等値比較の他に順序比較もできる
- 順序比較は先頭の文字からASCII表で比較
  - `abcd < abxa` → `true` . `c < x` であるため
  - `make < maker` → `true` . `4文字 < 5文字` であるため

```cpp
#include <iostream>

int main() {
  std::string x, ans{"C++"};
  std::cin >> x;

  if (x == ans) std::cout << " correct!\n";
  else if (x == "CPU") std::cout << " no...\n";
  else std::cout << " incorrect...\n";

  std::string a {"abc"};
  if (a < "xyz") std::cout << a <<" < xyz is true\n";
}
```

```
% ./a.out
C++
 correct!
abc < xyz is true
% ./a.out
CPU
 no...
abc < xyz is true
% ./a.out
hello
 incorrect...
abc < xyz is true
```

半開区間

- C++で良く使われる数学用語
- [a, b)を離散値で用いる（数学では実数が対象）

名前	表記法	要素の条件
閉区間	[a, b]	$a \leq x \leq b$
開区間	(a, b)	$a < x < b$
半開区間（左開右閉）	(a, b]	$a < x \leq b$
半開区間（左閉右開）	[a, b)	$a \leq x < b$

`string` 型変数内の文字へのアクセス

- 添字の利用（空文字列でない場合）
  - 配列の添字を使って変数内の文字にアクセス
  - n文字の文字列の添字は半開区間 `[0, n)`

メンバ関数	動作
`s.empty()`	`s` の長さが0であれば `true` を返す
`s.size()`	`s` の長さを返す
`s.front()`	`s` の先頭の文字を返す
`s.back()`	`s` の末尾の文字を返す

- メンバ関数（対象変数用の処理を行う関数）
  - `s` はstring型の変数とする

```cpp
#include <iostream>
int main() {
  std::string s {"abcXefg"};
  char ch = s[3]; // 右辺値として
  s[3] = 'd';     // 左辺値として
  std::cout << ch << " " << s << " " <<"\n";

  std::string t;
  std::cin >> t;
  if (!t.empty()) {   // 空文字列でない?
    std::cout << t[0] <<" "<< t[t.size()-1] <<" ";
    std::cout << t.front() <<" "<< t.back() <<" ";
    t.front() = 'X';  // 左辺値として
    t.back() = 'Z';   // 左辺値として
    std::cout << t <<"\n";
  }
}
```

```
% ./a.out
X abcdefg
hello
h o h o XellZ
```

`string` 型のメンバ関数

- `s` はstring型の変数とする

メンバ関数	動作
`s.find(x)`	`s` の先頭から `x` を探して添字を返す
`s.rfind(x)`	`s` の末尾から `x` を探して添字を返す
`s.substr(x, y)`	`s` の `x` 番目から `y` 個の文字（部分文字列）を返す
`s.replace(x, y, z)`	`s` の `x` 番目から `y` 個の文字を `z` に置き換える

- `find()`，`rfind()` で見つからない場合，特殊な値 `std::string::npos` が返却される

```cpp
#include <iostream>
int main() {
  std::string s {"abcdefghijabc"};
  int i, j, k;
  i = s.find("b");    // 前から見たbの添字
  j = s.find("cde");  // 前から見たcの添字
  k = s.rfind("abc"); // 後ろから見たaの添字
  std::cout << i << " " << j << " " << k << "\n";

  if (s.find("X") == std::string::npos)
    std::cout << "X is not found\n";

  std::cout << s.substr(2, 3) << "\n"; // cde
  std::cout << s.substr(9) << "\n";    // jabc

  s.replace(10, 3, "klmn"); // abcdefghijklmn
  std::cout << s << "\n";
}
```

```
% ./a.out
1 2 10
X is not found
cde
jabc
abcdefghijklmn
```

数値と文字列の相互変換

- `to_string(x)`：数値である `x` を数字の文字列へ変換
- `stoi(x)`：数字の文字列 `x` を `int` 値へ変換
- `stod(x)`：数字の文字列 `x` を `double` 値へ変換

※ 文字列の後ろ側には数字以外の文字が含まれていてもよい

```cpp
#include <iostream>

int main() {
  std::string s {"i = "};
  int i {10};
  s += std::to_string(i);

  double r {2.5};
  s += ", r = " + std::to_string(r);
  std::cout << s <<"\n";

  std::string ten{"10s"}, pi{"3.1415 rad"};
  i = std::stoi(ten);
  r = std::stod(pi);
  std::cout << i <<", "<< r <<"\n";
}
```

```
% ./a.out
i = 10, r = 2.500000
10, 3.1415
```

vector型

`vector`型

- 同一の型の値を複数同時に扱うためのデータ型
  - `char`も対象だが通常は`string`を使用する
- `<vector>`ヘッダファイルをインクルード
- 要素の型を宣言の際に指定する
- `n`文字の各要素は配列の添字を使ってアクセス
  - 半開区間`[0,n)`

```cpp
#include <iostream>
#include <vector>

int main() {
  std::vector<int>    x; // 初期化なしで宣言. int型.
  std::vector<double> y; // 初期化なしで宣言. double型.

  std::vector<int> b(3); // 要素3個, 初期値 0
  b[0] = 1;
  b[2] = 3;
  std::cout << b[1] <<" "
            << b[2] <<"\n"; // 0 3

  x = b; // 3要素のコピー
}
```

vectorの宣言

- 丸括弧と中括弧で指定の意味が異なる
- 初期値を指定すると型名を省略できる

```cpp
#include <iostream>
#include <vector>
int main() {

  std::vector<double> a;     // 要素数0

  std::vector<double> b(5);  // 要素数5, 初期値0.0

  std::vector<double> c(5, 1.4);  // 要素数5, 初期値1.4
  std::vector c2(5, 1.4);         // cの省略形

  std::vector<double> d {1.2, 2.5, 3.5};  // 中括弧の初期値指定
  std::vector d2 {1.2, 2.5, 3.5};         // dの省略形
}
```

要素の追加と削除

- `v.push_back( x )`: vector `v` の末尾に要素 `x` の追加
- `v.pop_back()`: vector `v` の末尾要素の削除
- `v.size()`: vector `v` の要素数
- `v.clear()`: vector `v` の全要素の削除

```cpp
#include <iostream>
#include <vector>

int main() {
  std::vector<double> x;
  x.push_back(1.5);
  x.push_back(2.8);
  x.push_back(3.3);
  std::cout << x.size() <<": " << x.front() <<" "<< x.back() <<"\n";

  x.pop_back();
  std::cout << x.size() <<": " << x.front() <<" "<< x.back() <<"\n";

  x.clear();
  std::cout << x.size() <<"\n";
}
```

```
% ./a.out
3: 1.5 3.3
2: 1.5 2.8
0
```

`vector` 変数の比較

- 先頭の要素からそれぞれ比較．すべて同じならば `true`，一つでも異なれば `false`．
- 各要素は比較可能でなければならない
  - ユーザ定義型の場合に注意

```cpp
#include <iostream>
#include <vector>

int main() {
  std::vector v1 { 1, 2, 3, 4, 5 };
  std::vector v2 { 0, 2, 3, 4, 8 };
  if (v1 == v2) std::cout <<"equal\n";
  if (v1 != v2) std::cout <<"not equal\n";
  if (v1 <  v2) std::cout <<"less than\n";
  if (v1 <= v2) std::cout <<"less than equal\n";
  if (v1 >  v2) std::cout <<"greater than\n";
  if (v1 >= v2) std::cout <<"greater than equal\n";
}
```

```
% ./a.out
not equal
greater than
greater than equal
```

`string`要素の`vector`

- 宣言時の省略に注意する
  - **文字列リテラルの初期値省略ではstring型とならない**
  - `std::string`が関係する時には`<string>`を省略しない
- 配列内の**文字**へのアクセス
  - 2段階で取り出す: `vs[0][1]` → `(vs[0])[1]`

```
1  #include <iostream>
2  #include <vector>
3  using std::vector, std::string, std::cout;
4  int main() {
5    vector<string> vs {"abc", "def", "ghi"};
6    // vector vs {"abc", "def", "ghi"}; // これはやらないほうが良い.
7                                        // std::vector<char const*>と解釈されてしまうため.
8
9    s = vs[0];
10   cout << s[1] <<" "<< vs[0][1] <<"\n"; // b b
11 }
```

`vector` の操作

メンバ関数	意味
`v.push_back(t)`	v の末尾に値tを追加する
`v.pop_back()`	v の末尾要素を削除する
`v.clear()`	v の全要素を削除する
`v[n]`	v の n 番目の要素
`v.size()`	v が持つ要素数を返す

メンバ関数	意味
`v.front()`	v の先頭要素
`v.back()`	v の末尾要素
`v.empty()`	v が空かどうか
`v1 = v2`	v1へのv2の上書き
`v1 == v2`	v1 と v2 が同一か
`v1 != v2`	v1 と v2 が同一でないか

pythonとの比較：`vector`

```cpp
// cppのプログラム
#include <iostream>
#include <vector>
int main() {
  std::vector<int> a;     // 要素数0
  std::vector<int> b(5);  // 要素数5, 初期値0
  std::vector<double> c(5, 1.4);  // 要素数5, 初期値1.4
  std::vector<double> d {1.2, 2.5, 3.5};  // 中括弧の初期値指定
  std::cout << d[0] <<" " << d[1] <<" "<< d[2] <<"\n";

  d.push_back(1.5);
  d.push_back(2.8);
  std::cout << d.size() <<": " << d.front() <<" "<< d.back() <<"\n";
  d.pop_back();
  std::cout << d.size() <<": " << d.front() <<" "<< d.back() <<"\n";
  d.clear();
  std::cout << d.size() <<"\n";

  vector<string> vs {"abc", "def", "ghi"};
  s = vs[0];
  std::cout << s[1] <<" "<< vs[0][1] <<"\n";
}
```

```python
# pythonのプログラム



a = []
b = [0] * 5 # 要素数5, 初期値0.0
c = [1.4] * 5 # 要素数5, 初期値1.4
d = [1.2, 2.5, 3.5] # 中括弧の初期値指定
print( d[0], d[1], d[2] )

d.append(1.5)
d.append(2.8)
print( len(d), ": ", d[0], " ", d[-1] )
d.pop()
print( len(d), ": ", d[0], " ", d[-1] )
d.clear()
print(len(d))

vs = ["abc", "def", "ghi"]
s = vs[0]
print( s[1], vs[0][1] )
```