C++プログラミングI

第3回 stringとvector

成蹊大学理工学部

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

- C 言語では char 型の配列 (C-string)
- 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
- string 型
  - 基本データ型ではなく標準ライブラリのデータ型
  - <string> ヘッダファイル

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

string: 初期化と代入

- 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
  - 型変換されている

```
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;            // "" 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
```

string: 連結

- +演算子は文字列の連結
- +=演算子で更新も可能

```
s3 = s2 + "! ";          // 文字列リテラル
s3 = s3 + s3;            // 変数
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
```

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）
- 可能：s + "abc" + "xyz"
- 可能："abc" + s + "xyz"
- 確認："abc" + "xyz" + s は？

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

```
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")
  cout << a <<" < xyz is true\n";
```

string 型変数内の文字

- 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)
- メンバ関数の利用
  - empty(), front(), back()

```
std::string s {"abcXefg"};
char ch = s[3]; // 右辺値として
s[3] = 'd';     // 左辺値として

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";
}
```

string 型のメンバ関数

- メンバ関数: 対象変数用の処理を行う関数
- find(): 先頭から探して添字を返す
- rfind(): 末尾から探して添字を返す
  - npos: 見つからない場合の特殊な値
- substr(): 部分文字列
- replace(): 部分文字列の置き換え

```
std::string s {"abcdefghijabc"};
int i, k;
i = 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
if (s.find("X") == std::string::npos)
  std::cout << "X is not found\n";
std::cout << s.substr(i, 3) <<"\n";
std::cout << s.substr(k) <<"\n";
s.replace(k, 3, "klmn");
std::cout << s << "\n";
```

数値と文字列の相互変換

- to_string(): 数値から数字の列へ
- stoi(): 数字の列から int 値へ
- stod(): 数字の列から double 値へ

```
std::string s;
int i {10};
double r {2.5};
s += std::to_string(i);
s += std::to_string(r);

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

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)

```
std::vector<int>    x;
std::vector<double> y;

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, C 言語配列では不可
```

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

```
std::vector<double> a;      // 要素数 0

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

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

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

要素の追加と削除

- push_back(): 末尾要素の追加
- pop_back() : 末尾要素の削除
- size(): 要素数
- clear(): 全要素の削除

```
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";
```

vector 変数の比較

- 先頭の要素からそれぞれ比較
- 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

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

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 段階で取り出すことになる
  - vs[0][1]→(vs[0])[1]

```
using std::vector, std::string, std::cout;

// 要素は string
vector<string> vs {"abc", "def", "ghi"};

// 要素は C-string
vector y {"abc","efg"};

s = vs[0];
cout << s[1] <<" "<< vs[0][1] <<"\n"; // b b
```

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
`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