Introduction to Using Standard Template Libraries

Speaker: Soichi Tanaka

今日話すこと

- C++ の STL について話します
- Java 勢のみなさん、ごめんなさい！
- これを使いこなせると解ける問題の幅が広がります

STL の概要

STL とは

- Standard Template Library の略
  - 型に応じて挙動を変える標準ライブラリ
  - 型指定のフォーマット: `func<型>`
- 今回説明するのは**コンテナ**と呼ばれるタイプ
  - コンテナ: 要素をたくさん持つデータのこと

```
#include <iostream>
#include <set>

using namespace std;

int main() {
  set<int> s;                   // int を要素に持つ集合 (s = {})
  s.insert(10);                 // 10 を追加 (s = {10})
  s.insert(20);                 // 20 を追加 (s = {10, 20})
  s.insert(10);                 // 10 を追加 (s = {10, 20})
  cout << s.size() << endl; // s の要素数 (= 2)
  return 0;
}
```

イテレータ

- コンテナの内部要素に直接アクセスする機能
- 内部要素を指すポインタのような存在
- 演算子はうまく定義されている
  - ++iter は次の要素へ

```
typedef set<int> S; // typedef で型に別名がつけられる

int main() {
  S s;
  for (int n = 0; n < 10; ++n) {
    s.insert(n*n - 2*n + 3);
  }
  // s.end() は終端を表す仮想的なイテレータ
  for (S::iterator iter = s.begin(); iter != s.end(); ++iter) {
    // *iter でその実体を表す
    // s の要素を列挙
    cout << *iter << endl;
  }
  return 0;
}
```

最低限覚えておきたい STL の種類

- `vector`: 可変長配列
- `set`: 集合
- `map`: 連想配列
- `queue`: キュー
- `stack`: スタック
- `priority_queue`: 優先度付きキュー

参考になるサイト

- http://www.cplusplus.com/reference/
- http://www.cppll.jp/cppreference/cpp_stl.html

vector - 可変長配列

- 配列みたいなやつ
- 配列の大きさを後から変えられる

```
#include <vector>

using namespace std;

int main() {
    vector<double> v;
    return 0;
}
```

vector の特徴

$\mathcal{O}(1)$ でできること

- ランダムアクセス
- 末尾要素の追加, 削除

$\mathcal{O}(n)$ かかってしまうこと

- 末尾以外の要素の追加, 削除
- 配列内に要素 x があるかの検索

メソッド

- v[i] ランダムアクセス
- v.push_back(x) 末尾へ追加
- v.pop_back() 末尾の削除
- v.size() 要素数

```
int main() {
  vector<int> v;
  for (int i = 0; i < 3; ++i) {
    v.push_back(10*i + 1); // 要素の追加
  }
  cout << "v: ";
  for (int i = 0; i < (int)v.size(); ++i) {
    cout << v[i] << " "; // 各要素の表示
  }
  while (v.size() > 0) {
    v.pop_back(); // 要素の削除
  }
  return 0;
}
```

その他

- vector はイテレータを使わない方が良いと思う
  - ソートするときだけイテレータを使う
- vector 同士の比較演算ができる
  - 辞書順比較みたいなことをする

```
#include <algorithm>  // sort
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int a[4] = {30, 10, 20, 40};
  vector<int> v1, v2;
  for (int i = 0; i < 4; ++i) v1.push_back(a[i]);
  v2 = v1;
  sort(v2.begin(), v2.end());
  for (int i = 0; i < (int)v2.size(); ++i) cout << v2[i] << endl;
  cout << (v1 >= v2? "v1 >= v2": "v1 < v2") << endl;
  return 0;
}
```

set - 集合

- 数学の集合と同じ
  - 要素が集合に含まれているか記憶する
  - 要素間の順序や各要素の個数は記憶されない
- イテレータを使うとソート順で要素を列挙できる
- 比較演算 < が未定義のものは集合にできない
  - 例えば複素数型 (complex)

```
#include <set>

using namespace std;

int main() {
    set<double> s;
    return 0;
}
```

set の特徴

$\mathcal{O}(\log n)$ でできること

- 要素の追加, 削除
- 集合内に要素 x が存在するかの判定

メソッド

- s.insert(x) 要素の追加
- s.count(x) 要素が含まれているか
  - s.find(x) != s.end() でもできる
- s.size() 要素数
- s.erase(x) 要素の削除

```
typedef set<int>::iterator Iter; // X::iterator は型なので別名がつけられる
int main() {
  set<int> s;    // s = {}
  s.insert(10); // s = {10}
  s.insert(20); // s = {10, 20}
  s.insert(10); // s = {10, 20}
  cout << s.size() << endl; // 2
  for (Iter iter = s.begin(); iter != s.end(); ++iter) {
    cout << *iter << endl; // ソート順 (10, 20 の順) で列挙
  }
  s.erase(10); // s = {20}
  cout << s.size() << endl; // 1
  return 0;
}
```

map - 連想配列

- 配列の拡張概念
- key を入力すると value にアクセスできるデータ型
  - (例) `height["Alice"] = 160`
    - key が `"Alice"` で value が `160`
- key の型は比較演算 `<` が定義されている必要がある
- イテレータを使うと key のソート順に列挙できる

```
#include <iostream>
#include <map>

using namespace std;

int main() {
  map<string, int> m; // map<key の型, value の型>
  return 0;
}
```

map の特徴

$\mathcal{O}(\log n)$ でできること

- ランダムアクセス
- 要素の追加と削除

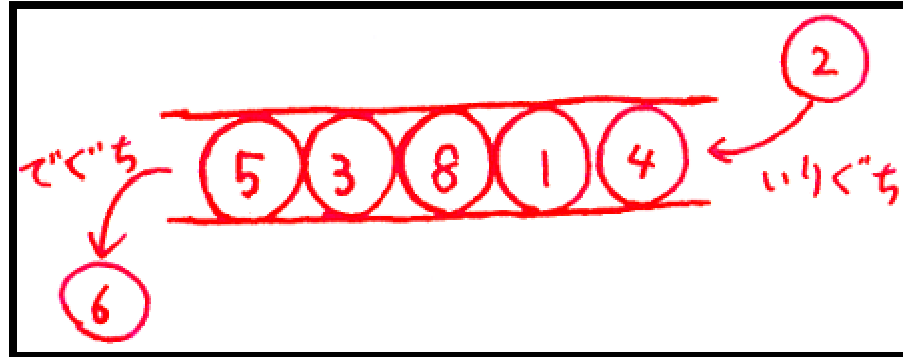
メソッド

- m.count(key) で要素の検索
- m[key] でランダムアクセスと要素の追加
- m.size() で要素数
- m.erase(key) で要素の削除

```
typedef map<string, int> M; // typedef で型に別名をつけられる

int main() {
  M m;                 // m = {}
  m["Alice"] = 160; // m = {"Alice": 160}
  m["Bob"] = 150;    // m = {"Alice": 160, "Bob": 150}
  m["Alice"] = 200; // m = {"Alice": 200, "Bob": 150}
  for (M::iterator iter = m.begin(); iter != m.end(); ++iter) {
    // key のソート順で列挙 ("Alice", "Bob" の順)
    // iter->first は key で iter->second は value
    cout << iter->first << ": " << iter->second << endl;
  }
  return 0;
}
```

queue - キュー



- FIFO (First In First Out)
  - 要素の追加と取り出しができる
  - 取り出し順は追加の**早い**順

```
#include <queue>

using namespace std;

int main() {
  queue<int> q;
  return 0;
}
```

queue の特徴

$\mathcal{O}(1)$ でできること

- FIFO に従ったデータの出し入れ

$\mathcal{O}(n)$ かかること

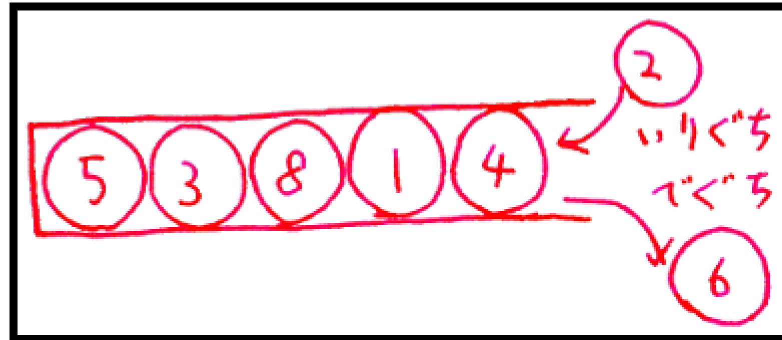
- FIFO に従わないランダムアクセス

メソッド (queue)

- q.pop() 先頭要素の削除
- q.front() 先頭要素の参照
- q.push(x) 要素の追加
- q.size() 要素数
- イテレータはない

```
int main() {
  int a[4] = {10, 30, 20, 30};
  queue<int> q;
  for (int i = 0; i < 4; ++i) q.push(a[i]);
  while (q.size() > 0) {
    cout << q.front() << endl; // 10, 30, 20, 30 の順
    q.pop();
  }
  return 0;
}
```

stack - スタック



- FILO (First In Last Out)
  - 要素の追加と取り出しができる
  - 取り出し順は追加の**遅い**順

```
#include <stack>

using namespace std;

int main() {
  stack<int> s;
  return 0;
}
```

stack の特徴

$\mathcal{O}(1)$ でできること

- FILO に従ったデータの出し入れ

$\mathcal{O}(n)$ かかること

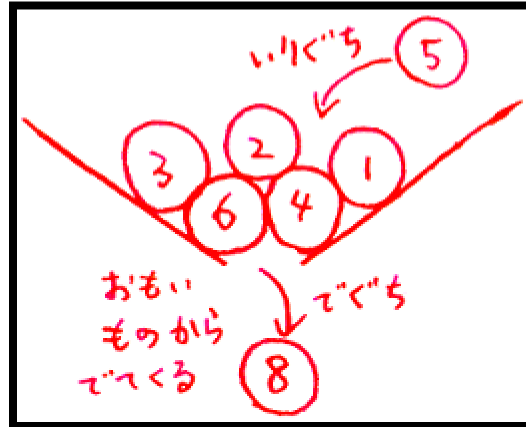
- FILO に従わないランダムアクセス

メソッド (stack)

- s.pop() 先頭要素の削除
- s.top() 先頭要素の参照
- s.push(x) 要素の追加
- s.size() 要素数
- イテレータはない

```
int main() {
  int a[4] = {10, 30, 20, 30};
  stack<int> s;
  for (int i = 0; i < 4; ++i) s.push(a[i]);
  while (s.size() > 0) {
    cout << s.top() << endl; // 30, 20, 30, 10 の順
    s.pop();
  }
  return 0;
}
```

priority_queue



- キューと大体同じだが、取り出しは最大の要素から
- 比較演算 < が定義されているもの限定

```
#include <queue>          // #include <priority_queue> ではない

using namespace std;

int main() {
  priority_queue<int> pq;
  return 0;
}
```

$\mathcal{O}(\log n)$ でできること

- 要素の追加
- 最大要素の取り出し

メソッド

- pq.pop() 最大要素の削除
- pq.push(x) 要素の追加
- pq.top() 最大要素の参照
- pq.size() 要素数
- イテレータはない

```
int main() {
  int a[5] = {10, 30, 20, 50, 20};
  priority_queue<int> pq;
  for (int i = 0; i < 5; ++i) pq.push(a[i]);
  while (pq.size() > 0) {
    cout << pq.top() << endl; // 50, 30, 20, 20, 10 の順
    pq.pop();
  }
  return 0;
}
```

その他

- 最小の要素から出るように変更することも可能

```
#include <iostream>
#include <queue>          // priority_queue
#include <vector>         // vector
#include <algorithm>      // greater
using namespace std;
int main() {
  // 最小要素から出てくるようにするおまじない
  priority_queue<int, vector<int>, greater<int> > pq;
  int a[5] = {10, 30, 20, 50, 20};
    for (int i = 0; i < 5; ++i) pq.push(a[i]);
    while (pq.size() > 0) {
        cout << pq.top() << endl; // 10, 20, 20, 30, 50 の順
        pq.pop();
    }
  return 0;
}
```

pair - ペア (おまけ)

- 2要素を持つ型
- 比較演算ができる
  - `first`, `second` の順で比較
- `pair(コスト, 本体データ)` が便利
  - priority_queue などで活躍

```cpp
#include <iostream>
#include <utility>

using namespace std;

int main() {
    pair<double, string> p(3.14, "pi");
    p.first = 3;
    cout << p.first << ", " << p.second << endl; // 3, pi
    return 0;
}
```

C++11 の話 その1

- C++11 よりも前のバージョンでは...
  - X<Y<Z> > のようにスペースを入れる必要がある
    - >> がシフト演算子だと判断されるため
- C++11 では X<Y<Z>> と書ける

```
int main() {
  // C++11 よりも前のバージョンでは
  // set<vector<int> > v; と書かなくてはならなかった
  set<vector<int>> v; // C++11 では OK
  return 0;
}
```

C++11 の話 その2

- コンテナへの代入が簡単に
- 配列の初期化みたいな書き方ができるようになった

```
int main() {
  // C++11 以前ではコンテナに対して
  // {} を使った初期化はできなかった
  vector<set<int>> v = {{3, 1, 2}, {2, 1, 2}};
  return 0;
}
```

C++11 の話 その3

range-based for loop が利用可能になった

- イテレータを持つコンテナで利用可能
- for (型 変数 : コンテナ) という構文

```
int main() {
  vector<set<int>> v = {{3, 1, 2}, {2, 1, 2}};
  for (set<int>& s : v) {   // v の各要素が順に s に代入される
    for (int x : s) {       // s の各要素が順に x に代入される
      cout << x << " ";     // 1 2 3 \n 1 2 \n
    }
    cout << endl;
  }
  return 0;
}
```

その他重要そうな STL

- deque: デック
  - vector の上位互換
  - 先頭と末尾に対する要素追加と削除が $\mathcal{O}(1)$
- list: リスト
  - vector 同様、要素の順序を記憶する
  - 短所: ランダムアクセスはできない
  - 長所: どんな位置でも要素挿入・削除が $\mathcal{O}(1)$
- unordered_set
- unordered_map
  - C++11 以降で利用可能
  - 長所: $\mathcal{O}(n \log n)$ 操作が $\mathcal{O}(1)$ (平均) に
  - 短所: ソート順ではなくなる

Thank you for your attention!