# C++プログラミングI

- 第14回：vectorやポインタを用いた間接参照
- 担当：二瓶芙巳雄

# 配列やポインタを用いた間接参照

- これまでの配列）主たる処理対象となるデータだけを配列の要素に
- 対象データを管理するための情報を配列にすることもできる
  - 配列の添字を要素とする配列
  - 配列要素へのポインタを要素とする配列
- 元の配列を全体集合と考えるならば、添字やポインタの配列はその部分集合を表すのに使用できる
- 部分集合は複数作ることができ、それらを通して間接的に元の全体集合の要素にアクセスする
- ある配列の要素の一部を別の配列にコピーするよりも、配列の添字やポインタを使って部分集合を表した方が都合の良い場合がある
  - 要素一つ一つのサイズが大きい
  - オリジナルを一ヶ所で管理したい
- 管理の仕方
  - 添字の数値を要素とする整数配列
  - 配列要素へのポインタの配列

# 日付の処理

- 日付の計算では月の取扱いがある
  - 1月、2月、3月,…, 12月
  - Jan, Feb, …, Dec
- 月の名前の配列、添字を月の数（1 → Jan）
- 添字0を無駄にしてよい
- `idx[]` を通して間接的に `montab[]` 配列の要素を指定

```
% ./a.out
Feb, Apr, Jun, Sep, Nov
Apr, May, Jun, Jul, Sep, Oct, Nov, Dec, Jan
Jan, Mar, May, Jul, Sep, Nov
```

```cpp
void print(const vector<int>& idx) {
  vector<string> montab {
    "", "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };

  for (size_t i = 0; i < idx.size(); i++) {
    if (i != 0) cout << ", ";
    cout << montab[ idx[i] ];  // 間接参照！
  }
  cout << "\n";
}

int main() {
  // 日数の少ない月
  vector<int> sm{2, 4, 6, 9, 11};
  print(sm);

  // 授業のある月
  vector<int> cl{4, 5, 6, 7, 9, 10, 11, 12, 1};
  print(cl);

  // 奇数月
  vector<int> om{1, 3, 5, 7, 9, 11};
  print(om);
}
```

# 日付の処理：間接参照を使わないと…

```cpp
void print(const vector<int>& idx) {
  vector<string> montab {
    "", "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };

  for (size_t i = 0; i < idx.size(); i++) {
    if (i != 0) cout << ", ";
    cout << montab[ idx[i] ];  // 間接参照！
  }
  cout << "\n";
}

int main() {
  vector<int> sm{2, 4, 6, 9, 11};
  print(sm);

  vector<int> cl{4, 5, 6, 7, 9, 10, 11, 12, 1};
  print(cl);

  vector<int> om{1, 3, 5, 7, 9, 11};
  print(om);
}
```

- `montab` の値が `{"", "睦月", "如月", "弥生", ..., "霜月", "師走" }` ;でも変更箇所が少ない

```cpp
void print(const vector<string>& month) {
  for (size_t i = 0; i < month.size(); i++) {
    if (i != 0) cout << ", ";
    cout << month[i];
  }
  cout << "\n";
}

int main() {
  vector<string> sm{"Feb", "Apr", "Jun", "Sep", "Nov"};
  print(sm);

  vector<string> cl{"Apr", "May", "Jun", "Jul",
    "Sep", "Oct", "Nov", "Dec", "Jan"};
  print(cl);

  vector<string> om{"Jan", "Mar", "May", "Jul", "Sep", "Nov"};
  print(om);
}
```

- わかりやすいコードではある
- 文字列のコピペをミスるだけでバグ混入
- 値を変えたいときとても面倒

# 辞書を用いたコード化

- 要素が重複して現れる場合の処理
- 重複を確認するための配列
- `25`, `20` や `1` という数字が何度か現れている。辞書を使ったコード化は、ファイルの圧縮、通信量の削減、暗号化といったさまざまな用途に応用可能

```
% ./a.out
THE RAIN IN SPAIN FALLS MAINLY ON THE PLAIN.
RAIN WATER, WATER THAT HAS FALLEN FROM THE CLOUDS IN RAIN.
TEARS ARE FALLING LIKE RAIN.
AFTER RAIN COMES FAIR WEATHER.
COME RAIN OR SHINE.
RAIN, RAIN, GO TO SPAIN.
```

```cpp
int main() {
  vector<string> dic{
    ",",     ".",      "AFTER", "ARE",    "CLOUDS",
    "COME",  "COMES",  "FAIR",  "FALLEN", "FALLING",
    "FALLS", "FROM",   "GO",    "HAS",    "IN",
    "LIKE",  "MAINLY", "ON",    "OR",     "PLAIN",
    "RAIN",  "SHINE",  "SPAIN", "TEARS",  "THAT",
    "THE",   "TO",     "WATER", "WEATHER"
  };

  vector<int> idx{
   25, 20, 14, 22, 10, 16, 17, 25, 19,  1, 20, 27,
    0, 27, 24, 13,  8, 11, 25,  4, 14, 20,  1, 23,
    3,  9, 15, 20,  1,  2, 20,  6,  7, 28,  1,  5,
   20, 18, 21,  1, 20,  0, 20,  0, 12, 26, 22,  1
  };


  for (size_t i = 0; i < idx.size(); i++) {
    string s{ dic[idx[i]] }; // 間接参照

    if (s == ",") cout << ",";
    else if (s == ".") cout << ".\n";
    else cout << " " << s;
  }
}
```

# ポインタによる検索用の索引

- 検索では結果が複数出てくる
- 検索対象の全体集合から部分集合を作り出す
- 添字配列の欠点
  - 元の配列と添字配列をセットで扱う必要がある
  - 関数の引数は必ず2つの配列を必要とする
- ポインタ配列の導入 → 要素ごとの間接参照
  - ポインタが必要な要素のアドレスを持っている
  - ポインタ配列のみで運用できる

```
1  // 添字配列を使う場合．引数が2つになる（1つ目：全体集合，2つ目：部分集合を示す添字）．
2  void print(vector<Data>& d, vector<int>& idx);
3
4  // ポインタを使う場合．引数は1つで良い（全体集合の任意の箇所をポインタは示せるため）．
5  void print(vector<Data*>& p);
```

# 検索プログラム

- `vector<Member>`：構造体の配列(変数`member`の型)
- `vector<Member*>`:ポインタの配列(変数`s`の型)
- 検索用の関数 `find_all()`
  - 検索結果である部分集合をポインタ配列として表す
  - `m`から検索し、要素のポインタを`plist`に設定
  - 戻り値の型`auto`は`return`文から決定される
- 一覧出力用の関数 `print()`
  - `p`は`Member*`型のため範囲`for`文で`&`は不要

```
% ./a.out
Honda 20
Suzuki 20

Okuda 30
Sakai 33
```

```
1  struct Member { std::string name; int age; };
2
3  auto find_all(vector<Member>& m, int age) {
4    vector<Member*> plist;
5    for (auto& x : m) // x の型は Member
6      if (age <= x.age && x.age < age + 10)
7        plist.push_back(&x);
8    return plist;
9  }
10
11 void print(const vector<Member*>& plist) {
12   for (auto p : plist) // p の型は Member*
13     std::cout << p->name << " " << p->age << "\n";
14   std::cout << "\n";
15 }
16
17 int main() {
18   vector<Member> members{
19     {"Kasai", 41},  {"Honda", 20}, {"Tanaka", 48},
20     {"Suzuki", 20}, {"Okuda", 30}, {"Sakai", 33} };
21
22   auto s {find_all(members, 20)};  // 20代を探す
23   print(s);
24   s = find_all(members, 30);  // 30代を探す
25   print(s);
26 }
```

# 要素数変更とアドレス

- `vector` に要素を追加するとメモリ位置が変わる
  - →「いつまでもそこにあると思うな」
- 以下はコンパイルできるが実行時に変になる

```
1  int main() {
2    std::vector<int> v;
3    std::vector<int*> vp;
4
5    for (int i{0}, x{0}; std::cin >> x; i++) {
6      v.push_back(x);
7      vp.push_back(&v[i]); // 意味的なエラー
8    }
9  }
```

`push_back()` は `v` の使用するメモリ領域を変更するので、ループが進むにつれて最初の方で保存したアドレスが無効になっていく

# 要素数変更とアドレス

- vector 要素のアドレスを利用する場合には、途中で要素数を変更をせずに、すべてが確定してからアドレスを使うように。

```cpp
//入力した値と間接的なポインタ
#include <iostream>
#include <vector>

int main() {
  std::vector <int> v;
  std::vector <int*> vp;

  for (int x; std::cin >> x; ) v.push_back(x); // ここでvを確定させる

  for (size_t i = 0; i < v.size(); i++) vp.push_back( &v[i] ); //要素のアドレスを取得
  // これ以降，v.push_back()すると，vの再配置が起き，vpが保存しているポインタが無効化される可能性がある

  for (auto p : vp) std::cout << *p < "\n";
  return 0;
}
```

# `vector<bool>` とアドレス

- **`std::vector<bool>` は特別な方法で実装されているので要素のアドレスが使えない**
  - `bool` 型は二値なので1ビットで表現可能
  - `vector<bool>` はメモリ使用量最小で実装されている
  - アドレス指定（例えば `&x[0]`）するとコンパイルエラー
- 制限：
  - 要素ごとのメモリアドレスは取得できない
  - 要素に対するリファレンスの指定もできない

```
1  int main() {
2    std::vector<bool> x {true, false, true, false};
3    std::cout << &x[1] <<"\n"; // コンパイルエラー
4    bool& a {x[2]}; // コンパイルエラー
5  }
```

# RGBファイルの検索

- Unix OSで使われるテキストファイル `rgb.txt`
  - ※ `rgb.txt` はCoursePower上のサンプルプログラム一式に含まれています
  - 光の三原色の情報
  - 各色が256段階で示された8ビットカラー
  - 色名は一個または複数個の英単語
- `Color` 構造体の配列にファイルの内容を読み込んで検索

```
1  struct Color { // 一色のデータ
2    int r, g, b;
3    string name;
4  };
```

`rgb.txt` の中身（抜粋）

```
255 250 250   snow
248 248 255   ghost white
248 248 255   GhostWhite
245 245 245   white smoke
245 245 245   WhiteSmoke
220 220 220   gainsboro
255 250 240   floral white
255 250 240   FloralWhite
253 245 230   old lace
253 245 230   OldLace
250 240 230   linen
250 235 215   antique white
250 235 215   AntiqueWhite
255 239 213   papaya whip
255 239 213   PapayaWhip
255 235 205   blanched almond
255 235 205   BlanchedAlmond
255 228 196   bisque
255 218 185   peach puff
255 218 185   PeachPuff
255 222 173   navajo white
255 222 173   NavajoWhite
...(後略)
```

# ソースコード全体

- 各部分を以降で説明
- 入力した色名に部分一致するものを表示

```
1   struct Color { int r, g, b; string name; };
2
3   auto read_file() {  // rgb.txt ファイルを読み込む
4     vector<Color> x;
5     std::ifstream fin("rgb.txt");
6
7     if (!fin) {
8       std::cerr << "read_file failed!\n";
9       return x;
10    }
11
12    // 1行に赤緑青色名がホワイトスペースで区切られて並ぶ
13    for (Color t;
14         fin >> t.r >> t.g >> t.b
15             >> std::ws && getline(fin, t.name);
16    ) x.push_back(t);
17
18    return x;
19  }
```

```
20  auto find_all(vector<Color>& rgb, string x) { // 検索
21    vector<Color*> clist;
22    for (auto& e : rgb)
23      if (e.name.find(x) != string::npos)
24        clist.push_back(&e);
25    return clist;
26  }
27
28  int main() {
29    auto rgb{read_file()};  // ファイルの読み込み
30
31    if (rgb.empty()) return 1;  // 読み込み失敗
32
33    // 検索用の色の名を入力
34    for (string cn; std::cin >> cn;) {
35      // 対象の絞り込み
36      for (auto p : find_all(rgb, cn)) {
37        std::cout << setw(4) << p->r << " "
38                  << setw(4) << p->g << " "
39                  << setw(4) << p->b << " "
40                  << p->name << "\n";
41      }
42    }
43    return 0;
44  }
```

# データの読み込み

- 赤緑青は0…255の整数
- `std::ws` でホワイトスペースを読み飛ばす
- 戻り値の型は `vector<Color>`

```
% cat rgb.txt
255 250 250   snow
248 248 255   ghost white
248 248 255   GhostWhite
245 245 245   white smoke
245 245 245   WhiteSmoke
...(後略)
```

```
1   struct Color { int r, g, b; string name; };
2
3   auto read_file() {  // rgb.txt ファイルを読み込む
4     vector<Color> x;
5     std::ifstream fin("rgb.txt");
6
7     if (!fin) {
8       std::cerr << "read_file failed!\n";
9       return x;
10    }
11
12    // 1行に赤緑青色名がホワイトスペースで区切られて並ぶ
13    // 式①「Color t;」 式②「fin >> t.r >> t.g >> t.b >> std::ws && getline(fin, t.name);」 式③なし
14    for( Color t; fin >> t.r >> t.g >> t.b >> std::ws && getline(fin, t.name); ) x.push_back(t);
15
16    return x;
17  }
```

# 色名の検索

- 範囲 `for` 文のリファレンス指定( `auto&` )は必須
  - → その要素のアドレスを登録することが目的
- 名前 `e.name` に `x` で指定した文字列を含むか
- 戻り値の型は `vector<Color*>`
- 検索結果である部分集合をポインタ配列として表す（複数要素の配列をそのまま返す）

```
1  // 検索
2  auto find_all(vector<Color>& rgb, string x) {
3    vector<Color*> clist;
4    for (auto& e : rgb) {
5      if (e.name.find(x) != string::npos) clist.push_back(&e);
6    }
7    return clist;
8  }
```

# `main()` の処理

- ファイルから `rgb` に読み込み
- 色名を `cn` に入力
- 条件に合う要素を一つずつ `p` に取り出す

```
% ./a.out
white
 248  248  255 ghost white
 245  245  245 white smoke
 255  250  240 floral white
 250  235  215 antique white
 255  222  173 navajo white
 255  255  255 white
black
   0    0    0 black
```

```cpp
int main() {
  // ファイルの読み込み
  auto rgb{read_file()};

  // 読み込み失敗
  if (rgb.empty()) return 1;

  // 検索用の色の名を入力
  for (string cn; std::cin >> cn;) {
    // 対象の絞り込み
    for (auto p : find_all(rgb, cn)) {
      std::cout << setw(4) << p->r << " "
                << setw(4) << p->g << " "
                << setw(4) << p->b << " "
                << p->name << "\n";
    }
  }
  return 0;
}
```

# 一時変数の省略

- `find_all()` の結果は `vector`
- 一時変数を利用した場合

```
1  auto clist { find_all(rgb, s) };
2  for (auto p : clist) {
3  ...
```

- 範囲 `for` 文で直接利用

```
1  for (auto p : find_all(rgb, s) ) {
2  ...
```

`find_all()` の結果がそのまま範囲for 文に使われる