Programming C++
Lecture Note 3
クラスの派生と継承

Jie Huang

クラスの派生と継承

- 派生クラスの定義

```
// 基底クラスcontainerの定義
class container {
public:
   int percent_loaded;
   container() {}
};
// containerの派生クラスboxの定義
class box : public container {
public: double height, width, lenght;
   box() {}
   double volume() { return height * width * length;}
};
```

クラスの派生と継承

- もう1つの基底クラス

```
int current_year = 2000;
class railroad_car {
public: int year_built; railroad_car(){}
  int age(){ return current_year-year_built;}
};
```

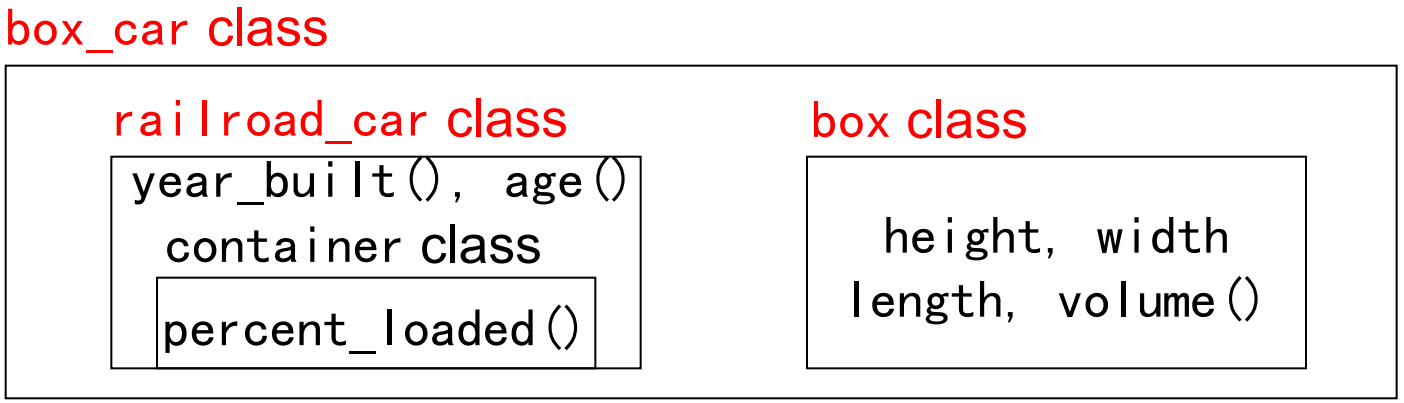
- 複数の基底クラスから派生したクラス

```
class box_car : public railroad_car, public box {
public:
  box_car(double h=3.5, double w=9.2, double l=40.0)
       { height = h; width = w; length = l; }
};
```

ここで、クラスbox_carはクラスrailroad_carとクラスbox両方の派生クラスである。同時に複数のクラスを基底クラスとすることを多重継承(multiple inheritance)という。

クラスの派生と継承

- public派生における継承(inheritance)
  public派生によって生じるクラスは基底クラスのメンバー変数やメンバー関数についてもアクセスできるので、派生クラスは基底クラスのメンバー変数とメンバー関数を含むことになる。このことを継承という。



- もし、派生クラスと基底クラスに同じ名前のメンバー変数がある場合は、基底クラスのメンバー変数が隠蔽され、アクセスできなくなる。
- クラススコープ演算子を使うことによって、明示的に参照することが可能でなる。例えば、
  box::volume(); //boxクラスのvolume関数が呼び出される

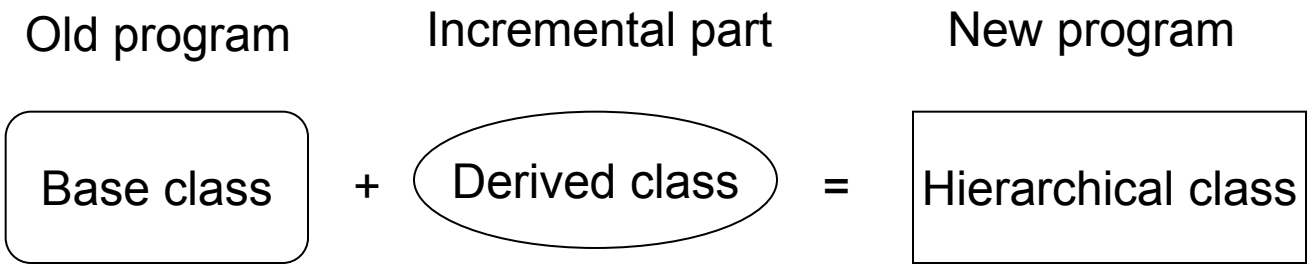
クラスの派生と継承

- 派生のアクセス指定が継承に与える影響

基底クラスにおける変数と関数	public派生クラスでのアクセス制限	protectedクラスでのアクセス制限	privateクラスでのアクセス制限
public	public	protected	Private
protected	protected	protected	Private
private	No access	No access	No access

クラスの派生と継承

- クラス派生によってもたらされる効果
  - 機能の追加
    新しい機能をクラスの派生によって追加すること、差分プログラミング(incremental programming)ともいう。
  - パッケージソフトの従来のインタフェースを残しつつ、新しい機能を加える時などに利用できる。



クラスの派生と継承

- 基底クラスに共通の処理をまとめる
  2つ以上の異なるクラスの共通の部分をまとめることによって、重複機能を減らし、共通のインタフェースを提供できる。例えば、box_carクラスにおける基底クラスrailroad_car、container、boxなどのような使い方。
- 一般化と特殊化の構造をつくる
  基底クラスとほぼ同じ性質のクラスを派生クラスを使うことによって基底クラスの特殊化を行うことができる。例えば、以下のflat_carクラスはコンストラクタ以外は自分自身のメンバー変数もメンバー関数もなく、box_carの特殊クラスとして定義できる。

```
class flat_car : public box_car {
  public:
    flat_car(double w=9.2, double l=40.0):box_car(8.25,w,l){}
 };
```

クラス階層(hierarchy)設計の心得

- 自然のカテゴリを反映する
  ここでいうカテゴリとは実在のものでもよければ、抽象的なものでもよい。例えば、鉄道車両のクラス定義のように実在のものによりクラス定義を行う。
- 重複定義を避ける
  そのためには、クラスの概念をよく整理すること。メンバー変数や関数はより広い範囲でつかわれるかもしれないので、他のクラスからも利用できるようにする。
- 頻繁な重複演算を避ける
  例えば、鉄道車両クラスにおける円筒の定義はradiusによるものとdiameterによるもの、の二通りが考えられる。どちらが頻繁に使われるのかによって、定義の選択を行う。
- 抽象化を行う
  クラスのオープンインタフェースを定義し、内部変数や関数などはprivate部におき、隠蔽を行う。

プログラミングする工夫

- 有効桁数を指定して出力する
```
streamsize prec = cout.precision();//現在の設定をprecにセーブ
cout << setprecision(3) << output_data //有効桁3のデータ出力
     << setprecision(prec) << endl; //設定を元に戻す
```
- 改良版、小数点の位置を固定し、出力の長さをそろえる
```
cout.setf(ios::fixed); //設定を小数点以下の桁数にする
cout << setprecision(1) //小数点以下の桁数を1とする
cout << setw(7) << output_data; //幅7文字でデータを出力
```
- 注）以上の設定を有効にするためには
<iomanip>をインクルードする必要がある。
なお、コンパイラーはg++を推奨する

プログラミングする工夫

- ポインタ変数とconst指定
  - char型へのconstポインタ

    char *const c = “abc”;

    ポインタの値は変更できないが、ポインタの指す内容は変更できる。

    c[0] = ‘x’; // 正しい

    c = “xyz”; // できない
  - const char型へのポインタ

    const char *c = “abc”;

    ポインタの値は変更できるが、ポインタの指す内容は変更できない。

    c = “xyz”; // 正しい

    c[0] = ‘x’; // できない
  - const char型へのconstポインタ

    const char *const c = “abc”;

    この場合、ポインタの値もポインタの指す内容も変更はできないことになる。

vectorコンテナを使う

- vectorの定義
  - vectorはtemplateによって定義されるので、中身のタイプをパラメーターで指定することができる。
    ```
    vector<double> homework; //要素がdouble型変数のvector
    ```
  - vectorは配列のようなものであるが、データを追加したり、取り出したり、便利な機能が備わっている。
    ```
    while (cin >> x)
      homework.push_back(x); //vectorに要素を後ろから追加する
    ```
  - typedefを使って定義を簡単にする
    ```
    // vectorクラスで用意されるデータサイズを表す変数型を
    // typedefを使って簡単にする
    typedef vector<double>::size_type vec_sz;
    // すっきりとした変数定義ができる
    vec_sz size = homework.size();
    ```

vectorコンテナを使う

- vectorの要素を挿すイテレータ
  - イテレータはポイントのようなものであるが、クラスによって定義され、より複雑な処理が可能である。
  - vectorの最初の要素へのイテレータは
    homework.begin();
  - vectorの最後の要素の次へのイテレータは
    homework.end();
    注意：最後の要素ではなく、最後の次の要素を指すイテレータで、そこには要素は存在しない。
  - vectorの真中の要素median

```
vec_sz mid = size/2;
double median = size % 2 == 0 ?
    (homework[mid]+homework[mid-1])/2
    : homework[mid];
```

vectorコンテナを使う

- vectorの要素をソートする
  - vectorをソートするには単に
    sort(homework.begin(), homework.end());
    ここで、ソートの範囲は半分オープンの
    [homework.begin(), homework.end())となる。
- C++標準ライブラリの関数としてのvectorへの性能要求
  - vectorへの要素の追加は計算量が0（n）より悪くない
  - vectorのソートは計算量が0（nlog（n））より悪くない

プログラム例1

```
#include <iomanip>
#include <iostream>
#include <string>
using namespace std;
int main(){
  cout << “Please enter your first name: ”;
  string name; cin >> name;
  cout << “Hello, ” << name << “!” << endl;
  cout << “Please enter your midterm and final exam grades: ”;
  double midterm, final; cin >> midterm >> final;
  cout << “Enter all your homework grades, ”
          “followed by end-of-file: ”;
  int count = 0; double sum = 0; double x;
  while (cin >> x) {  ++count;   sum += x; }
  streamsize prec = cout.precision();
  cout << "Your final grade is " << setprecision(3)
    << 0.2 * midterm + 0.4 * final + 0.4 * sum / count
    << setprecision(prec) << endl;
  return 0;
}
```

プログラム例2

```
#include <algorithm>
#include <iomanip>
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main(){
  cout << "Please enter your first name: ";
  string name; cin >> name;
  cout << "Hello, " << name << "!" << endl;
  cout << "Please enter your midterm and final exam grades: ";
  double midterm, final; cin >> midterm >> final;
  cout << "Enter all your homework grades, “
          "followed by end-of-file: ";
  vector<double> homework; double x;
  while (cin >> x) homework.push_back(x);
```

プログラム例2

```
typedef vector<double>::size_type vec_sz;
vec_sz size = homework.size();
if (size == 0) {
  cout << endl << "You must enter your grades.  “
                  "Please try again." << endl;
  return 1;
}
sort(homework.begin(), homework.end());
vec_sz mid = size/2;
double median;
median = size % 2 == 0 ?
  (homework[mid]+homework[mid-1])/2 : homework[mid];
streamsize prec = cout.precision();
cout << "Your final grade is " << setprecision(3)
     << 0.2 * midterm + 0.4 * final + 0.4 * median
     << setprecision(prec) << endl;
return 0;
}
```