ソフトゼミC
第3回 C++の基礎(続き)

エレクトロニクス研究部

2014/08/04

今日は何をするの？

- オーバーロードとは
- new/delete演算子
- コンストラクタのオーバーロード
- クラスの継承

オーバーロード(1)

- オーバーロードとは

同じ名前の関数を複数定義すること
引数の型が違う場合に対応する
例えば、値を2倍にして返す関数を作る
この時引数がint型かdouble型かわからない
→両方の型に対応させたい

オーバーロード(2)

- オーバーロードの例

```
#include<iostream>
using namespace std;
int dbr(int a){
    return a*2;
}
double dbr(double a){
    return a*2;
}
```

オーバーロード(3)

- オーバーロードの例(続き)

```
int main(){
        int a =10;
        cout << a << "*2=" <<dbr(a) << endl;
        double b = 20.25;
        cout << b << "*2=" << dbr(b) << endl;
        return 0;
}
```

- 出力結果

```
10*2=20
20.25*2=40.5
```

オーバーロード(4)

- 解説

同じ関数でも、引数か戻り値のどちらかが違っていれば複数宣言できる。

このように１つの名前を複数の目的に対応させることをポリモーフィズムと言う

ポリモーフィズムはカプセル化と並んで

オブジェクト指向の重要な考え方のひとつである。

練習問題1

- 値を読み込み、2乗した値を返す関数pow()を作れ。ただしint型、double型、long型に対応させること
- 答え

```
#include<iostream>
using namespace std;
    int pow(int a   ){ return a*a; }
double pow(double a){ return a*a; }
   long pow(long a  ){ return a*a; }
```

練習問題1

- 答え(続き)

```
int main(){
	int a =10;
	cout << a << "^2=" <<pow(a) << endl;
	double b = 20.25;
	cout << b << "^2=" << pow(b) << endl;
	long c = 30;
	cout << c << "^2=" << pow(c) << endl;
	return 0;
}
```

new演算子(1)

- メモリの割り当て

C言語ではmalloc()を使ってメモリの割り当てを行なっていたが、C++ではnew演算子を使う

例）int型のポインタに実態を持たせる

```
int *p;
p = new int;
*p = 10;
cout << "*p=" << *p << endl;
```

→ *p=10

new演算子(2)

- classに対してnew演算子を使う

```
#include<iostream>
using namespace std;
class Test{
    int a;
public:
    Test(int x){ a=x; }
    void show(){ cout << a << endl; }
};
```

new演算子(3)

- classに対してnew演算子を使う

```
int main(){
	Test *p;
	p = new Test(10);
	p->show();
	return 0;
}
```

main文のnew演算子を消すとコンパイルこそ通るものの実行時に処理が停止するか、変な値が出る。

これは実体の無いものを強引に参照しようとしたため

delete演算子

- 割り当てたメモリを解放する

先ほどの例で割り当てた*pを解放すると

```
Test *p;
p = new Test(10);
p->show();
delete p;
```

Testクラスにデストラクタがあると

メモリの解放時に読み出される。

コンストラクタのオーバーロード(1)

- コンストラクタのオーバーロード

先ほどやったオーバーロードは

コンストラクタにも適用出来る。

大きな目的としてはコンストラクタの引数がある場合とない場合両方に対応することが

上げられる。

コンストラクタのオーバーロード(2)

- 例(23.cpp)

```
#include<iostream>
using namespace std;
class Test{
public:
	Test(){
		cout << "引数はありません" << endl;
	}
	Test(int x){
		cout << "引数は" << x << "です。" << endl;
	}
};
```

コンストラクタのオーバーロード(3)

- 例続き(23.cpp)

```
int main(){
	Test a;
	Test b(20);
	return 0;
}
```

→引数はありません

引数は20です。

1つ目のTestクラスには引数がないので引数なしのコンストラクタが呼ばれる、2つ目のTestクラスでは引数があるので引数のあるコンストラクタが呼ばれる。

練習問題2

- 昨日の練習問題4で作った人物の名前、年齢を要素としてもつクラスPersonを作る。
- この時に引数がない時、引数が名前だけの時、年齢だけの時、両方あるときを想定して

  コンストラクタを複数作れ
- ヒント

名前を受け取る引数は(char *str)で受け取る

strcpy(a,b);を使ってbを先頭ポインタとする文字列をaを先頭ポインタにする文字列にコピーする。

→答えは例によって長いの別紙に(24.cpp)

クラスの継承(1)

- 元のクラスをベースに新しいクラスを作りたいとき、クラスの継承というのを行う
- 継承はクラスの宣言時に書き、形式は

class クラス名 : アクセス指定子 継承するクラス名

と書く、アクセス指定子によって親クラスのメンバーがどれだけ使えるかが変わってくる

クラスの継承(2)

- アクセス指定子と、親クラスへのアクセスについて

	継承時のアクセス指定子			
親クラスでのアクセス指定子		private	protected	public
	private	アクセス不可	アクセス不可	アクセス不可
	protected	private	protected	protected
	public	private	protected	public

- protectedは継承するときに意味を成す
- privateと同じようにクラスの内部からしか参照できない

クラスの継承(3)

- 実際に継承をしてみる。(25.cpp)

```
#include<iostream>
using namespace std;
class base{
	int x;
public:
	void set(int a){ x = a; }
	void show(){ cout << "x = " << x << endl; }
	int getx(){ return x; }
};
```

クラスの継承(4)

- 例続き(25.cpp)

```
class derived :public base{
        int y;
public:
        void dset(int a){ y = a; }
        void show(){
                cout << "x = " << getx() << endl;
                cout << "y = " << y << endl;
        }
};
```

クラスの継承(5)

- 例続き(25.cpp)

```
int main(){
        base a;
        a.set(10);
        a.show();
        derived b;
        b.set(20);
        b.dset(30);
        b.show();
        return 0;
}
```

クラスの継承(6)

- 解説(25.cpp)

このように親クラスのprivateメンバには直接
アクセスできない。
どうしてもアクセスしたい場合は値を返すメンバ
関数を作るか、protectedにすること
また、親クラスにも子クラスにも両方同じメンバ
関数があると、子クラスの方が優先される。
このように親クラスと子クラスで違う動作をさせる
ことをオーバーライドと言う

継承時のコンストラクタ(1)

- 継承を使った時のコンストラクタの動き

```
#include<iostream>
using namespace std;
class B1{
public:
        B1(){ cout << "親クラスのコンストラクタが呼ばれました" << endl; }
        ~B1(){ cout << "親クラスのデストラクタが呼ばれました" << endl; }
};
class D1 : public B1{
public:
        D1(){ cout << "子クラスのコンストラクタが呼ばれました" << endl; }
        ~D1(){ cout << "子クラスのデストラクタが呼ばれました" << endl; }
};
```

継承時のコンストラクタ(2)

- 継承を使った時のコンストラクタの動き

```
int main(){
        D1 b;
        return 0;
}
```

→親クラスのコンストラクタが呼ばれました
　子クラスのコンストラクタが呼ばれました
　子クラスのデストラクタが呼ばれました
　親クラスのデストラクタが呼ばれました

継承時のコンストラクタ(3)

- コンストラクタに引数があるとき

```
#include<iostream>
using namespace std;
class B1{
        int x;
public:
        B1(int a){
                x = a;
        }
        int getx(){
                return x;
        }
};
```

継承時のコンストラクタ(4)

- コンストラクタに引数があるとき

```
class D1 : public B1{
        int y;
public:
        D1(int a,int b):B1(a){
                y = b;
        }
        void show(){
                cout << "x:" << getx() << endl << "y:" << y <<endl;
        }
};
```

継承時のコンストラクタ(5)

- コンストラクタに引数があるとき

```
int main(){
        D1 b(10,20);
        b.show();
        return 0;
}
```

→x:10
　y:20

子クラスのコンストラクタから親クラスのコンストラクタに値を渡すときには

D1(int a,int b):B1(a){ ...　のように書く

多重継承(1)

- C++では複数のクラスを継承することができる。

A,B,Cという3つのクラスがあるとき

A→B→Cというように順に継承することも

CがAとBの両方を継承することもできる。

- 例(28.cpp)

```
#include<iostream>
using namespace std;
```

多重継承(2)

- 例(28.cpp)

```
class B1{
public:
        B1(){ cout << "B1のコンストラクタ" << endl; }
        ~B1(){ cout << “B1のデストラクタ" << endl; }
};
class B2{
public:
        B2(){ cout << "B2のコンストラクタ" << endl; }
        ~B2(){ cout << “B2のデストラクタ" << endl; }
};
```

多重継承(3)

- 例(28.cpp)

```
class D1 : public B1,public B2{
public:
        D1(){ cout << "D1のコンストラクタ" << endl; }
        ~D1(){ cout << "D1のデストラクタ" << endl; }
};
int main(){
        D1 *p = new D1;
        cout << endl;
        delete p;
        return 0;
}
```

THE END

1日目お疲れ様でした。
次はJavaScriptについてやっていきます。