Prof. Dr. O. Rheinbach
Dipl.-Math. F. Röver
Institut für Numerische Mathematik und Optimierung
TU Bergakademie Freiberg

5th Exercise in HPC

Exercise 1
Consider the following program to recall the concept of inheritance and late binding (runtime polymorphism) in object oriented programming languages.

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

class BaseClass
{
public:
    virtual void Name();
};

void BaseClass::Name()
{
    cout<<"I am of type BaseClass."<<endl;
}

class DerivedClass:public BaseClass
{
public:
    void Name();
};

void DerivedClass::Name()
{
    cout<<"I am of type DerivedClass."<<endl;
}

int main()
{
    BaseClass bc;
    DerivedClass dc;

    bc.Name();
    dc.Name();
    (BaseClass(dc)).Name(); /* downcast: casting down to base is possible */
    BaseClass *p;     // a pointer to BaseClass...
    p=&bc;            // can point to bc...
    (*p).Name();
    p=&dc;            // but also to dc
    (*p).Name();

    int i=0;
    cout<<"Which?"<<endl;
    cin>>i;
    cout<<"You chose "<<i<<endl;

    if(i==1)
        p=&bc;
    else
        p=&dc;

    (*p).Name();
}
```

What happens if you downcast `*p` to BaseClass in the last line?

Exercise 2
The so-called CRTP is a technique to implement compile-time polymorphism (avoiding the overhead of runtime polymorphism). Try to emulate the example given above using the CRTP technique.

```
template<class DerivedClass>
class BaseClass
{
    // methods within BaseClass can use template parameter
    // to access members of DerivedClass
};
class DerivedClass : public BaseClass<DerivedClass>
{
    // ...
};
```