

```

#ifndef _SAHRK_H
#define _SAHRK_H

//    Shark.h
//    G. Volger
//    1.23.01
//    Shark class

#include <fstream.h>
#include "apstring.h"

class Shark
{
    public:
        // Constructors
        Shark();
        Shark(const apstring & name);
        Shark(const apstring & name, int age);
        Shark(const Shark & another);           // Copy constructor

        // Accessors
        void Print(ostream & out) const;
        int GetAge() const;
        apstring GetName() const;

        // Mutators
        void Input(istream & in);
        const Shark & operator += (const Shark & rhs);

    private:
        apstring myName;
        int myAge;
};

// Can we add an integer value to a Shark? - Why do we need two different functions?
// Why can't this function be a member function?
Shark operator + (int growth, const Shark & rhs);
// What can't we return a constant reference? We get the following error - why?
// Warning : function result is a pointer/reference to an automatic variable
// shark.cpp line 60 return temp;

Shark operator + (const Shark & lhs, int growth);

// Cannot add a Shark to a Shark producing a third Shark!
// Can we add a Shark to an existing Shark?
// What would you have to do to add a Shark to a Shark, producing a new Shark?

// Relational operator - based on Shark's age
bool operator < (const Shark & lhs, const Shark & rhs);
bool operator == (const Shark & lhs, const Shark & rhs);
bool operator > (const Shark & lhs, const Shark & rhs);

// I/O with Sharks
ostream & operator << (ostream & out, const Shark & jaws);
istream & operator >> (istream & in, Shark & jaws);

#endif

```

```

#include "shark.h"
#include <fstream.h>

// shark.cpp
// G. Volger
// 1.23.01
// A Shark class

Shark::Shark() : myName(""), myAge(0)
{
}

Shark::Shark(const apstring & name) : myName(name), myAge(0)
{
}

Shark::Shark(const apstring & name, int age) : myName(name), myAge(age)
{
}

Shark::Shark(const Shark & another)
{
    myName = another.myName;
    myAge = another.myAge;
}

void Shark::Print(ostream & out) const
{
    out << "" << myName << " (" << myAge << ")";
}

int Shark::GetAge() const
{
    return myAge;
}

apstring Shark::GetName() const
{
    return myName;
}

void Shark::Input(istream & in)
{
    in >> myName >> myAge;
}

const Shark & Shark::operator += (const Shark & rhs)
{
    Shark lhs(*this); // Need copy constructor to do this
    myAge = lhs.myAge + rhs.myAge;
    return *this;
}

Shark operator + (int growth, const Shark & rhs)
{
    int newAge = rhs.GetAge() + growth;
    Shark temp(rhs.GetName(), newAge); // Need a temp because we can't change rhs
    return temp;
}

```

```

Shark operator + (const Shark & lhs, int growth)
{
    return growth + lhs;                // Use the function already defined
}

bool operator < (const Shark & lhs, const Shark & rhs)
{
    return (lhs.GetAge() < rhs.GetAge());
}

bool operator == (const Shark & lhs, const Shark & rhs)
{
    return (lhs.GetAge() == rhs.GetAge());
}

bool operator > (const Shark & lhs, const Shark & rhs)
{
    return (rhs < lhs);                // < is already defined
}

ostream & operator << (ostream & out, const Shark & jaws)
{
    jaws.Print(out);                   // Do we really need to have a Print
    return out;                         // member function?
}

istream & operator >> (istream & in, Shark & jaws)
{
    jaws.Input(in);                    // Do we really need to have a Input
    return in;                          // member function?
}

```