

```
//=====
// SListInt.h
//=====
// Declarations for SListInt

#ifndef SLIST_INT_H
#define SLIST_INT_H

class SListInt {
    //-----
    // Constructor and destructor
    //-----

public:
    SListInt();
    ~SListInt();

    //-----
    // Copy constructor, swap, assignment operator
    //-----

public:
    SListInt(const SListInt& lst);
    void      swap(SListInt& lst);
    SListInt& operator=(const SListInt& lst);

    //-----
    // Member functions
    //-----

public:
    void push_front(int v);
    int  size() const;
    int  at(int idx) const;
    int& at(int idx);
    void reverse_v1();
    void reverse_v2();
    void print() const;

    //-----
    // Private member functions and fields
    //-----

private:
    struct Node {
        int  value;
        Node* next_p;
    };

    Node* m_head_p;
};

#endif /* SLIST_INT_H */
```







```

//=====
// types-dpoly.h
//=====
// Declarations for misc types

#ifndef TYPES_DPOLY_H
#define TYPES_DPOLY_H

//-----
// IObject
//-----

class IObject {
public:
    virtual ~IObject(){};
    virtual IObject* clone() const = 0;
    virtual bool eq(const IObject& rhs) const = 0;
    virtual bool lt(const IObject& rhs) const = 0;
    virtual void print() const = 0;
};

bool operator==(const IObject& lhs, const IObject& rhs);
bool operator<(const IObject& lhs, const IObject& rhs);

//-----
// Integer
//-----

class Integer : public IObject {
public:
    Integer();
    Integer(int data);

    Integer* clone() const;
    bool eq(const IObject& rhs) const;
    bool lt(const IObject& rhs) const;
    void print() const;

private:
    int m_data;
};

//-----
// Double
//-----

class Double : public IObject {
public:
    Double();
    Double(double data);

    Double* clone() const;
    bool eq(const IObject& rhs) const;

```

```
    bool    lt(const IObject& rhs) const;
    void    print() const;

private:
    double m_data;
};

//-----
// Complex
//-----

class Complex : public IObject {
public:
    Complex();
    Complex(double real, double imag);

    Complex* clone() const;
    bool    eq(const IObject& rhs) const;
    bool    lt(const IObject& rhs) const;
    void    print() const;

private:
    double m_real;
    double m_imag;
};

#endif // TYPES_DPOLY_H
```

```

//=====
// types-dpoly.cc
//=====
// Definitions for misc types

#include "types-dpoly.h"

#include <cstdio>

//-----
// Overloaded operators
//-----

bool operator==(const IObject& lhs, const IObject& rhs) {
    return lhs.eq(rhs);
}

bool operator<(const IObject& lhs, const IObject& rhs) {
    return lhs.lt(rhs);
}

//-----
// Integer
//-----

Integer::Integer() {
    m_data = 0;
}

Integer::Integer(int data) {
    m_data = data;
}

Integer* Integer::clone() const {
    /*' SECTION TASK '-----
    // Implement clone
    //-----
}

bool Integer::eq(const IObject& rhs) const {
    const Integer* rhs_p = dynamic_cast<const Integer*>(&rhs);
    if (rhs_p == nullptr)
        return false;
    else
        return (m_data == rhs_p->m_data);
}

bool Integer::lt(const IObject& rhs) const {

```

```

    const Integer* rhs_p = dynamic_cast<const Integer*>(&rhs);
    if (rhs_p == nullptr)
        return false;
    else
        return (m_data < rhs_p->m_data);
}

void Integer::print() const {
    std::printf("%d", m_data);
}

//-----
// Double
//-----

Double::Double() {
    m_data = 0.0;
}

Double::Double(double data) {
    m_data = data;
}

Double* Double::clone() const {
    //''' SECTION TASK '''
    // Implement clone
    //'''

}

bool Double::eq(const IObject& rhs) const {
    const Double* rhs_p = dynamic_cast<const Double*>(&rhs);
    if (rhs_p == nullptr)
        return false;
    else
        return (m_data == rhs_p->m_data);
}

bool Double::lt(const IObject& rhs) const {
    const Double* rhs_p = dynamic_cast<const Double*>(&rhs);
    if (rhs_p == nullptr)
        return false;
    else
        return (m_data < rhs_p->m_data);
}

void Double::print() const {
    std::printf("%f", m_data);
}

```



```

//=====
// SListIObj.h
//=====
// Declarations for SListIObj

#ifndef SLIST_IOBJ_H
#define SLIST_IOBJ_H

#include "types-dpoly.h"

class SListIObj {
    //-----
    // Constructor and destructor
    //-----
public:
    SListIObj();
    ~SListIObj();

    //-----
    // Copy constructor, swap, assignment operator
    //-----
public:
    SListIObj(const SListIObj& lst);
    void      swap(SListIObj& lst);
    SListIObj& operator=(const SListIObj& lst);

    //-----
    // Member functions
    //-----

public:
    void      push_front(const IObject& v);
    int       size() const;
    IObject*  at(int idx) const;
    IObject*& at(int idx);
    void      reverse_v1();
    void      reverse_v2();
    void      print() const;

    //-----
    // Private member functions and fields
    //-----

private:
    struct Node {
        IObject* obj_p;
        Node*    next_p;
    };

    Node* m_head_p;
};

#endif /* SLIST_IOBJ_H */

```

```
//-----  
// SListIObj Default Constructor  
//-----  
SListIObj::SListIObj() {  
    /* SECTION TASK #4 */  
    // Implement the constructor  
    /*  
}   
  
//-----  
// SListIObj Destructor  
//-----  
SListIObj::~SListIObj() {  
    /* SECTION TASK #4 */  
    // Delete the node  
    /*  
}   
  
//-----  
// SListIObj::push_front  
//-----  
void SListIObj::push_front(const IObject& v) {  
    /* SECTION TASK #4 */  
    // push to the front  
    /*  
  
}
```

```
//-----  
// SListIObj::size  
//-----  
int SListIObj::size() const {  
    /* SECTION TASK #5 */  
    // Implement the constructor  
    /*  
}   
  
//-----  
// SListIObj::at  
//-----  
IObject* SListIObj::at(int idx) const {  
    /* SECTION TASK #5 */  
    // Implement the constructor  
    /*  
}   
  
//-----  
// SListIObj::at  
//-----  
IObject*& SListIObj::at(int idx) {  
    /* SECTION TASK #5 */  
    // Implement the constructor  
    /*  
}   
  
}
```