C++ Abstract Base Classes

- **Abstract base classes permit complete separation of interface and implementation... -** This allows selection of an implementation for an interface at run-time. Benefits: (1) run-time flexibility for the user, (2) development flexibility for the developer, (3) near idealic encapsulation. For example, limited or no recompilation is required to introduce new implementations. **Note:** inheritance for specialization comes later....
- **at the price of run-time overhead. -** Each method in the interface is actually invoked through a pointer to the method in the implementation class. The size of this overhead in a system depends on the compiler, the frequency of use of objects of a class in a system, and other factors. "Typical" = 3%-10%
- **Polymorphism -** Given a pointer to an abstract base class, one call invoke methods through that pointer without knowing which implementation is actually being used "behind the scenes". Framework's Abstract Event class can work with BaBar's and CDF's different and evolving event implementations.

Intermediate C++
18-26 August 1997
Rob Kennedy
Liz Sexton-Kennedy

Trybos Abstract Record

- **Object Statement: Trybos Abstract Record**
  A Trybos record is an ordered collection of Trybos "banks". It can read its contents to a Trybos file or write it to a Trybos file. Clients will search for one or all banks matching a bank "key" = (name, number). Clients will need to use a variety of implementations with different I/O versus bank search CPU efficiency characteristics. Bank data may be event data, or calibration data, or....
- **Design Statement: This is an abstract class candidate**
- **Basic Abstract Class Form: Similar to concrete classes**
  - **Constructors same:** Base class init irrelevant since no base init to be done
  - **Destructor declared virtual:** virtual ~Class(void) ;
  - **All other member methods declared pure virtual:**
    virtual bool is_valid(void) const = 0 ; \\ virtual and "= 0' means "pure virtual"
  - **Friend methods require special treatment:** Use a member method to represent interface, implement friends in child classes with member method implementation. So, implement "friend bool operator ==(item1, item2)" in terms of member "bool compare(item2)" in implementation class.

TRY_Abstract_Record...


```
\\... Omitting some stuff...
class TRY_Abstract_Record
{ public:
TRY_Abstract_Record(void) ;
virtual ~TRY_Abstract_Record(void) ;
virtual int size(void) const = 0 ;                          \\ pure virtual method
virtual bool append(const TRY_Generic_Bank& bank) = 0 ;     \\ pure virtual method
protected:
TRY_IO_Buffer _iobuf ;    \\ Optimization: Data member shared by ALL implementation classes
private:
TRY_Abstract_Record(const TRY_Abstract_Record& record) ;    \\ Never an implementation!
TRY_Abstract_Record& operator =(const TRY_Abstract_Record& record) ;
friend bool operator ==(const TRY_Abstract_Record& record1,
                        const TRY_Abstract_Record& record2) ;    \\ also operator !=()
} ;
```

- **Methods removed from basic class form:** declared "private"
- **A feature common to all implementations:** A data member has "percolated up" to this base class after implementations compared.

And an Implementation...

Intermediate C++
18-26 August 1997
Rob Kennedy
Liz Sexton-Kennedy

```
\\... Omitting some stuff...
class TRY_Simple_Record: public TRY_Abstract_Record
{ public:
TRY_Simple_Record(void) ;                                 \\ Defined in compiled source file
virtual ~TRY_Simple_Record(void) ;                        \\ Defined in compiled source file
virtual int size(void) const    { return(_iobuf[0] + 1) ; }
virtual bool append(const TRY_Generic_Bank& bank) ;   \\ Defined in compiled source file
protected:
\\ Nothing...
private:
TRY_Simple_Record(const TRY_Simple_Record& record) ;      \\ Never an implementation!
TRY_Simple_Record& operator =(const TRY_Simple_Record& record) ;
friend bool operator ==(const TRY_Simple_Record& record1,
                        const TRY_Simple_Record& record2) ;    \\ also operator !=()
} ;
```

- **Lack of inlined methods OK:** Remember the costs of virtual methods [Meyers1]. CPU efficiency not priority (difficult w/virtual anyway).
- **"Redeclare" private elements of basic class form:** Implementation class may be used directly! Maintain the parent's interface.

And Its Use.

```
\\ BEGIN:  DRIVER CODE
TRY_Abstract_Record *p_record ;
TRY_Abstract_File       *p_file ;
p_file       = new  TRY_Disk_File ;         \\ Choose file implementation HERE
p_record = new TRY_Simple_Record ; \\ Choose record implementation HERE
\\ Check for new failures, open file, check for open failure, ...

\\ BEGIN: USER ROUTINE, arguments are (p_file, p_record)
while (p_file->read(*p_record))             \\ *p_record is detail of interface
  { cout << "Record size = " << p_record->size() << endl ;
  }
\\ END: USER ROUTINE

delete p_record ;                                   \\ Clean-up record implementation
delete p_file ;                                     \\ cleanup file implementation, too
\\ END: DRIVER  CODE
```

☞ **User Routine just works:** Whichever implementation is chosen for TRY_Abstract_Record (or for TRY_Abstract_File). User or developer can select amongst released implementations (with caveats). I/O efficiency for simple analysis, data search efficiency for reconstruction. Same user code.

Intermediate C++
18-26 August 1997
Rob Kennedy
Liz Sexton-Kennedy

More Refinements

☞ **New implementation for TRY_Abstract_Record ?**
Just "new" an instance of new class in the driver code, or use the same name as an old implementation class (newer version replaces older, no recompilation).

☞ **Express "forced" commonality in abstract base class**
Do not try to anticipate commonality too early, though. Let it "percolate up" during development as understanding of the classes/product improves.

☞ **Implementation classes serving multiple abstract base classes**: Probably a design weakness for novices, but may be useful. This may require the use of the inheritance qualifier ***virtual***. Suppose a Pion_Candidate inherits from both Meson and Track. Suppose both of these inherit from Candidate. Without virtual inheritance, there are ***TWO*** Candidates in Pion_Candidate. With virtual inheritance, there is only ***ONE***. Declare:
class Pion_Candidate: virtual public Meson, virtual public Track

Summary: Abstract Classes

☞ **Abstract classes represent extensible, polymorphic interfaces.** They "do" nothing without implementation classes. They can, however, "contract for" many implementations, at expense of CPU efficiency.

☞ **Start from basic abstract class form, then refine as with concrete classes (const, explicit, constants, etc.)**

☞ **Test with prototype implementation classes.**

☞ **Later, factor out common or required functionality amongst implementations, put it into the abstract base class ("percolate the commonality upwards")**

☞ **Inlining:** Still may be useful documentation style, but not a CPU-saver.

Commentary

Intermediate C++
18-26 August 1997
Rob Kennedy
Liz Sexton-Kennedy

☞ **Declare *private* that part of the "basic class form" which you explicitly do not wish to implement.**

☞ **kennedy@fnal.gov (Rob K)**
☞ **sexton@fnal.gov (Liz S-K)**