Sets for C++: A Pedagogical and Powerful Tool

Juan Alvarez Rubio
Dept. de Ciencias de la Computacion - U. de Chile
Blanco 2120, Santiago, Chile - jalvarez@dcc.uchile.cl
Dept. de Ingenieria Telematica - U. Politecnica de Madrid
Ciudad Universitaria, Madrid 28040, Spain - jalvarez@dit.upm.es

Abstract

C++ is a superset of the C language that, maintaining the compatibility, overcomes many shortcomings and extends it with features to write object-based programs (classes, overloading, templates), and object-oriented programs (inheritance, dynamic binding, polymorphism).

The implementation of a generic Set ADT represents a good synthesis of the C++ object-based features. The interface design, that extends the Pascal Model based on operator overloading, can be implemented through an array, a characteristic vector and a linked list. The Set ADT can be applied in several problem domains: sieve of Eratosthenes, text analysis, enumerations.

1 Introduction

The sets are present from basic to higher education. In computer science, the interest is the mathematical concept [Tucker92] and also the computational representation of sets [Aho92]. For example, Pascal is usually taught in the first computing course, including operations with sets [Reilly90]. However, the main pedagogical interest is the consideration of sets as an abstract data type (ADT) [McCracken87]. Then, it is important to define a user interface and to implement the operations using distinct representations without effects on client programs. Thus, the importance of the ADT concept is clearly shown through a well-known, useful and important example.

Not all the imperative languages provide features to implement and use ADTs. Standard Pascal has not them, so the extended versions Turbo and Think Pascal are usually taught. The Pascal limitation was overcome in its successor Modula-2. The C language makes possible the implementation of ADTs through files and static and external variables. The stack development in [Kernighan88] illustrates this alternative.
The C++ programming language [Ellis90, Lippman92, Stroustrup91] provides a qualitatively higher level for ADTs implementation. It allows to mimic, almost completely, the potentiality of the pre-defined types, achieving the goal of the user defined types. To the class, the mechanism to encapsulate the ADT interface and representation, is added the overloading of function and operators (around 50). Additionally, the templates permit to define generic ADTs. So, there are comfortable and convenient features.

The Ada language has similar features (packages, overloading, generic), so this can be considered an object-based language [Wegner92]. However, the inheritance and dynamic binding facilities do C++ an object-oriented language [Wegner92]. Ada-9x will also have an inheritance mechanism.

In sum, the C++ implementation of the Set ADT is a good demonstration of object manipulation. In order to evaluate the C++ language for educational or production purposes, you can read a more complete analysis [Alvarez93].

2 Discussion

The development of the Set ADT will be introduced following the sections:
- interface design
- implementation
- applications

2.1 Interface Design

Ideally, the Set ADT interface is the Mathematics language. There is no programming language that provides the natural interface for sets. So, we will analyze some existent alternatives.

2.1.1 Pascal Sets

The standard Pascal language implements many operations on sets through a convenient operator notation:

<table>
<thead>
<tr>
<th>operation</th>
<th>example</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>union</td>
<td>$A + B$</td>
<td>set</td>
</tr>
<tr>
<td>intersection</td>
<td>$A \cap B$</td>
<td>set</td>
</tr>
<tr>
<td>difference</td>
<td>$A - B$</td>
<td>set</td>
</tr>
<tr>
<td>member</td>
<td>$x \in A$</td>
<td>boolean</td>
</tr>
<tr>
<td>equal</td>
<td>$A = B$</td>
<td>boolean</td>
</tr>
<tr>
<td>unequal</td>
<td>$A &lt;&gt; B$</td>
<td>boolean</td>
</tr>
<tr>
<td>subset</td>
<td>$A &lt;= B$</td>
<td>boolean</td>
</tr>
<tr>
<td>superset</td>
<td>$A &gt;= B$</td>
<td>boolean</td>
</tr>
<tr>
<td>declaration</td>
<td>var $A$, $B$: set of 1..100;</td>
<td></td>
</tr>
<tr>
<td>construction</td>
<td>[1,2,3] or [1..3]</td>
<td></td>
</tr>
<tr>
<td>assignment</td>
<td>$A := [1,2,3]$;</td>
<td></td>
</tr>
</tbody>
</table>
The next program, that determines the prime numbers between 1 and n using the sieve of Eratosthenes method, illustrates the application of the Pascal features:

```pascal
program Main;
  const n = 100;
  var Primes, Sieve: set of 1..n;
    i, j: integer;
begin
  Primes := [1];
  Sieve := [2..n];
  i := 2;
  while Sieve <> [] do begin
    if i in Sieve then begin
      Primes := Primes + [i];
      j := i;
      while j <= n do begin
        Sieve := Sieve - [j];
        j := j + i
      end;
    end;
    i := i + 1
  end;
  for i:=1 to n do
    if i in Primes then
      write(i)
end.
```

2.1.2 Functional notations

In some imperative programming languages the set operations must be implemented through subprograms, therefore, a functional notation is needed (for example Modula-2 [McCracken87]). In C++, there are implementations through functions [Stroustrup91][Coplien92] and combining functions and some assignment operators [Lea91].

A functional interface has the inconvenience when it is successively applied. For example, the expression \( C \leftarrow (A \cup B) - (A \cap B) \) would be translated in the following way:

```c++
//result in the third operand: f(set X, set Y, set& Z)
union(A, B, C);
inter(A, B, D);
minus(C, D, C);
//result in the first operand: f(set& X, set Y)
assign(C, A);  //o C = A
```
union (C, B);
assign(D, A);
inter (D, B);
minus (C, D);

//result in the class object: class set {...f(set X); ...}
C.assign(A);
C.union (B);
D.assign(A);
D.inter (B);
C.minus (D);

//result as the function value: set f(set X, set y);
C = minus( union(A,B), inter(A,B) )

2.1.3 Proposed Interface

The C++ language permits to overload the predefined operators, so we can use and extend the Pascal model associating a proper semantics to the symbols with an intuitive and natural meaning for sets. The following table shows the mapping between the mathematical and the C++ notation:

<table>
<thead>
<tr>
<th>operation</th>
<th>Mathematics</th>
<th>C++</th>
<th>result type</th>
</tr>
</thead>
<tbody>
<tr>
<td>union</td>
<td>A ∪ B</td>
<td>A + B</td>
<td>set</td>
</tr>
<tr>
<td>intersection</td>
<td>A ∩ B</td>
<td>A * B</td>
<td>set</td>
</tr>
<tr>
<td>difference</td>
<td>A − B</td>
<td>A − B</td>
<td>set</td>
</tr>
<tr>
<td>symmetric difference</td>
<td>AΔB</td>
<td>A ~ B</td>
<td>set</td>
</tr>
<tr>
<td>complement</td>
<td>A'</td>
<td>~A</td>
<td>set</td>
</tr>
<tr>
<td>member</td>
<td>x ∈ A</td>
<td>x &lt; A</td>
<td>boolean</td>
</tr>
<tr>
<td>not member</td>
<td>x ∉ A</td>
<td>!(x &lt; A)</td>
<td>boolean</td>
</tr>
<tr>
<td>equal</td>
<td>A = B</td>
<td>A == B</td>
<td>boolean</td>
</tr>
<tr>
<td>unequal</td>
<td>A ≠ B</td>
<td>A != B</td>
<td>boolean</td>
</tr>
<tr>
<td>subset</td>
<td>A ⊆ B</td>
<td>A &lt;= B</td>
<td>boolean</td>
</tr>
<tr>
<td>proper subset</td>
<td>A ⊂ B</td>
<td>A &lt; B</td>
<td>boolean</td>
</tr>
<tr>
<td>superset</td>
<td>A ⊇ B</td>
<td>A &gt;= B</td>
<td>boolean</td>
</tr>
<tr>
<td>proper superset</td>
<td>A ⊃ B</td>
<td>A &gt; B</td>
<td>boolean</td>
</tr>
<tr>
<td>cardinal</td>
<td>#A</td>
<td>+A</td>
<td>int</td>
</tr>
<tr>
<td>assignment</td>
<td>A ← B</td>
<td>A = B</td>
<td>set</td>
</tr>
<tr>
<td>insert an element</td>
<td>A ← A ∪ {x}</td>
<td>A += x</td>
<td>set</td>
</tr>
<tr>
<td>remove an element</td>
<td>A ← A − {x}</td>
<td>A -= x</td>
<td>set</td>
</tr>
<tr>
<td>display</td>
<td></td>
<td>cout &lt;&lt; A</td>
<td>ostream</td>
</tr>
</tbody>
</table>

The expression to determine the symmetric difference between two sets can be written in the following two ways:

C = A+B - A*B;
C = A ~ B;
Note that the first statement does not need parenthesis. The overloading means to keep the syntax (arity), the associativity, and the precedence of the operators. A functional notation is also allowed:

```cpp
operator=(C, operator ~(A, B)); // friend operators
C.operator=(A, operator ~(B)); // member operators
```

The declaration below is the C++ version of the generic Set ADT for elements of a type T with values between MinT and MaxT:

```cpp
#include <iostream.h>
enum Boolean{false, true};

template<class T, int MinT, int MaxT>
class Set {
public:
    Set operator+(const Set& A) const; // Union
    Set operator*(const Set& A) const; // Intersection
    Set operator-(const Set& A) const; // Difference
    Set operator~(const Set& A) const; // Symmetric difference
    Set operator~() const; // Complement
    Boolean operator==(const Set& A) const; // Equals?
    Boolean operator!=(const Set& A) const; // Unequal?
    Boolean operator<=(const Set& A) const; // Subset?
    Boolean operator<=(const Set& A) const; // Proper Subset?
    Boolean operator>=(const Set& A) const; // Superset?
    Boolean operator>(const Set& A) const; // Proper Superset?
    int operator() const; // cardinal
    Set& operator=(const Set& A); // Assignment
    Set& operator=(const T x); // insert x in Set
    Set& operator~=(const T x); // remove x from Set
    int operator() const; // cardinal
friend Boolean operator<(const T x, const Set<T, MinT, MaxT>& A); // Member?
friend ostream& operator<<(ostream& os, const Set<T, MinT, MaxT>& A);
// Constructors and Destructor
    Set(); // {} (default: empty Set)
    Set(const T x); // {x}
    Set(const T min, const T max); // {min, min+1, ..., max}
    Set(const T array[]); // {array[0], array[1], ... }
    Set(const Set& A); // copy constructor
~Set(); // destructor
private:
...
};
```
The line `template<class T, int MinT, int MaxT>` defines the class parameters. The first one requires a predefined type (or another class) and the last ones constant integer values. A class (or struct) contains data and functions, specified as public or private. The public members, usually functions, constitute the class interface for client programs.

Two of the operators (`<`, `<<`) are defined as friends of the class because of the first operand is not a class object. In the membership operator the first operand is an element, and in the "display" operator is an ostream.

The components of the private part (omitted to make easy the explanation) can not be used outside the class. They are reserved to define the internal and hidden ADT representation.

### 2.2 Class Implementation

Generally, there are more than one alternative to implement an ADT. The internal representation must be decided, and consequently the body of the operations must be written. The client programs only use the public interface and not depend of the chosen representation. Thus, a representation change does not affect the program, although C++ requires recompilation.

In the special case of sets, there are at least three classical representations: array, characteristic vector, linked list. In order to analyze and compare, we will show the private part, a constructor, and the union operator.

There are some conventions to write C++ ADTs. The public interface of the class usually appears in a file with the name `<class_name>.h`. This file also contains the private part in order to inform the compiler the size and the structure of the objects. However, the functions with their bodies usually appear in the file `<class_name>.c`. This file must contain the preprocessor statement `#include <class_name>.h` to access the public and private part of the class.

#### 2.2.1 Array implementation

The well-known representation of a set through an array can be complemented with the cardinal of the set. Thus, many operations will be more efficient (the cardinal itself for example). So, the private part of the class is:

```c
const int MAX = MaxT - MinT + 1; // maximum cardinal
short n; // cardinal: 0..MAX
T a[MAX]; // elements: {a[0], ..., a[n-1]}
```

The members `MAX`, `n` and `a` can only be used in the function bodies, but these are hidden and inaccessible to client programs. In any case, all the arrays need space for `MAX = MaxT - MinT + 1` variables of type `T`.

A constructor is a function with the same name of the class. It is executed when an object is defined. For example, the constructor that initializes a set with one element is:
template<class T, int F, int L> inline
Set<T,F,L>::Set(const T x) {
    if(F <= x && x <= L){
        n = i;
        a[0] = x;
    }
}

Note that all the functions must be defined using a template with the same
parameters of the class (although different names can be used). Also, the rela-
tion function/class is specified with the prefix Set<T,F,L>:: before the function
or operator name. The keyword inline is recommended to suggest the compiler
an in-inline expansion of the called function in order to avoid the overhead.
This mechanism replace and correct the parameterized macros (#define) of the
C language preprocessor.

On the other hand, the + operator can be implemented in the following way:

template<class T, int F, int L> inline
Set<T,F,L> Set<T,F,L>::operator+(const Set<T,F,L>& A) const{
    Set<T,F,L> temp = *this;
    for(int i=0; i < A.n; ++i)
        temp += A.a[i];
    return temp;
}

The local object declaration Set<T,F,L> temp = *this; includes an initialization
with other object of the same class. In fact, this is a pointer to the first
operand, therefore *this represents the operand itself. The constructor that
receives an object of the same class (copy-constructor) admits the alternative
invocation Set<T,F,L> temp(*this);

Finally, it is possible to implement an operation using others. For example,
the + operation uses the insertion of an element. The inline expansion avoids
eventual inefficiencies.

2.2.2 Characteristic vector implementation

C++ is one of the few languages that allows an effective and efficient imple-
mentation of sets through a characteristic vector. In this case, each possible
element has a bit. The bit means that the element is (or not) a member of the
set. Thus, a very compact representation is reached, that needs as bits as the
size of the universe.

Efficiency reasons recommend to limit the universe size to the number of
bits of a long int. Thus, the sets operations can be implemented through the
bit handling operators available in C++. Then, the private part will be:
long Bits; //characteristic vector
const int MAX = 8*sizeof(long); //maximum cardinal

The C++ sizeof operator gives the size in bytes for one object of the type. On the other hand, the restriction of the set size can be overcome defining an array of $MAX = (MaxT - MinT + 1)/(8 \times sizeof(long))$ long elements.

The constructor that initialize a set with one element is:

```cpp
template<class T, int F, int L> inline
Set<T,F,L>::Set(const T x) {
    Bits = 1 << (x-F);
}
```

The expression $1 << (x-F)$ produces a binary pattern with zeros, and 1 at the position corresponding to the element x. $a << b$ gives a result corresponding to a left shift of a in b bits positions.

The union of two sets is a bitwise OR between the internal representations:

```cpp
template<class T, int F, int L> inline
Set<T,F,L> Set<T,F,L>::operator+(const Set<T,F,L>& A) const{
    Set<T,F,L> temp;
    temp.Bits = Bits | A.Bits;
    return temp;
}
```

### 2.2.3 Linked list implementation

In order to overcome the storage consumption of the array alternative, and the universe size restriction of the characteristic vector, a linked list (ordered in our case) is usually utilized. Then, the private part is a pointer to a recursive structure:

```cpp
struct TE { T value; TE *next; };
TE *head;
```

Consequently, the constructor must ask dynamic storage for the element. The new operator gives the address of a piece of memory for one element:

```cpp
template<class T, int F, int L> inline
Set<T,F,L>::Set(const T x) {
    if (F<=x && x<L){
        head = new Set<T,F,L>::TE;
        head->value = x;
        head->next = NULL;
    }
}
```
The union operation can be carried out copying the first set and adding each element of the second set (using the corresponding operator):

```cpp
template<class T, int F, int L> inline
Set<T,F,L> Set<T,F,L>::operator+(const Set<T,F,L>& A) const{
    Set<T,F,L> temp = *this;
    for(Set<T,F,L>::TE* p=A.head; p!=NULL; p=p->next)
        temp += p->value;
    return temp;
}
```

This implementation requires a destructor (a function with the name `~<Set>`) Therefore the destructor is implicitly called before releasing the memory occupied by an object. Considering that C++ lacks an automatic garbage collector, the dynamic memory of the set elements must be returned.

```cpp
template<class T, int F, int L> inline
Set<T,F,L>::~Set(){
    Set<T,F,L>::TE *p, *q;
    for(p=head; p!=NULL; p=q){
        q = p->next;
        delete p;
    }
}
```

The delete operator sends back the memory of the pointed object. New and delete are operators, so they can be overloaded in user defined classes.

### 2.2.4 Comparison of implementations

The following table shows the output of the UNIX word count (wc) command for the files with the implementation of the operators:

<table>
<thead>
<tr>
<th>implementation</th>
<th>lines</th>
<th>tokens</th>
<th>characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>149</td>
<td>548</td>
<td>3619</td>
</tr>
<tr>
<td>characteristic vector</td>
<td>136</td>
<td>521</td>
<td>3475</td>
</tr>
<tr>
<td>linked list</td>
<td>206</td>
<td>717</td>
<td>5198</td>
</tr>
</tbody>
</table>

The relation between numbers coincides with the difficulties found to implement the operations. However, the characteristic vector implementation restricts the set size to the bits on a long int.

### 2.3 Applications

The generic set ADT can be applied in several problems. The users programs must be indistinctly executed with any representation.
Three programs were selected with criteria of simplicity, shortness, usefulness and pedagogy: sieve of Eratosthenes, consonants in a text, operations on sets of an enumerated type.

All client program of an ADT must include the corresponding X.h file. It means that the program must be compiled with the public interface and the private representation of an object.

### 2.3.1 Sieve of Eratosthenes

In order to prove the three representations, the problem has been limited to determine the prime numbers between 1 and 30:

```c
main() {
    const int n = 30;
    Set<short,1,n> Primes(1), Sieve(2,n), Empty;

    for(short i=2; Sieve != Empty; ++i)
        if( i < Sieve ){
            Primes += i;
            for(int j = i; j <= n; j += i)
                Sieve -= j;
        }
    cout << Primes;
}
{1,2,3,5,7,11,13,17,19,23,29}
```

The program is notably more compact that the corresponding Pascal program, making evident the writing easiness of the C++ language (inherited from the base language C). Notice that the output, generated by the `<<` operator, shows the sets in the natural and ordered notation of sets.

The curious notation `Set<short,1,n>` to declare sets is due to the instantiation of the generic ADT for sets of shorts with values between 1 and n.

The definition of a class object invokes a constructor (a function with the name of the class). Thus, `Primes(1)` calls a constructor that receives an integer and initializes the set with the element 1. On the other hand, `Sieve(2,n)` invokes a constructor with two parameters creating a set with the elements 2, 3, ..., n. A constructor without parameters (default constructor) generates an empty set.

The following table shows the time performance (the output of the UNIX time command) of the program with different values of n:

<table>
<thead>
<tr>
<th>implementation</th>
<th>n=30</th>
<th>n=100</th>
<th>n=1000</th>
<th>n=10000</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>0.0</td>
<td>0.0</td>
<td>0.4</td>
<td>40.9</td>
</tr>
<tr>
<td>characteristic vector</td>
<td>0.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>linked list</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
<td>24.9</td>
</tr>
</tbody>
</table>
2.3.2 Text Consonants

Sets are usually utilized in order to analyze the letters in a text, because a unique copy of each letter is often needed. For example, to determine the consonants letters in a text file, the following program can be applied:

```
//Consonants in the standard input
#include <ctype.h>
main()
{
    Set<char, 'a', 'z'> Alphabet('a', 'z'), Vowels("aeiou"), C;
    char c;
    while( cin.get(c) )
    {
        if (c=tolower(c)) < Alphabet - Vowels )
            C += c;
        cout << C;
    }
}
```

The generic ADT is instantiated for sets of lower case letters. Vowels("aeiou") calls a constructor that receive an array with elements of the same type of the set members.

With a file of 29940 characters, the following times were registered:

<table>
<thead>
<tr>
<th>implementation</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>array</td>
<td>3.2</td>
</tr>
<tr>
<td>characteristic vector</td>
<td>0.9</td>
</tr>
<tr>
<td>linked list</td>
<td>13.4</td>
</tr>
</tbody>
</table>

2.3.3 Operations on sets of an enumerated type

C++ admits rudimentary enumerations representing integer constants. For example, the next two statements are equivalents:

```cpp
enum {false,true};
const int false=0, true=1;
```

The enumerations considerably increase the readability. The following program shows sets whose elements are of an enumerated type:

```cpp
enum day {Mo=1,Tu,We,Th,Fr, Sa, Su};
ostream& operator<<(ostream& os, const day d){
    static char* s[]={"Mo","Tu","We","Th","Fr","Sa","Su"};
    return os << s[d-Mo];
}
ostream& operator<<(ostream& os, const Boolean b){
    static char* s[]={"false","true"};
    return os << s[b.false];
}
```
main()
{
    Set<day,Mo,Su> A(Mo, Th), B(We, Su);
    cout << "A = " << A << " A = " << A + B << " \n";
    cout << "A = B = " << A * B << " \n";
    cout << "A ~ B = " << A ^ B << " \n";
    cout << "A <= B = " << (A <= B) << " \n";
    cout << "A >= B = " << (A >= B) << " \n";
    cout << "Mo < A = " << (Mo < A) << " \n";
    cout << "¬A = " << ¬A << " \n";
    cout << "+A = " << +A << " \n";
}

The parenthesis are needed because the << operator has higher precedence than some other operators. On the other hand, the << overloading allows to show the results symbolically and not numerically:

A = {Mo, Tu, We, Th}  B = {We, Th, Fr, Sa, Su}
A + B = {Mo, Tu, We, Th, Fr, Sa, Su}
A * B = {We, Th}
A - B = {Mo, Tu}
A ^ B = {Mo, Tu, Fr, Sa, Su}
A == B = false
A != B = true
A <= B = false
A >= B = false
Mo < A = true
¬A = {Fr, Sa, Su}
+A = 4

3 Conclusions

The C++ implementation of the generic Set ADT represents a pedagogical and powerful tool. Powerful, because it translates the mathematical set concept. Pedagogical, because it shows and summarizes the design, implementation, and application issues involved in an ADT development. At the same time, an educational space is created to experiment, measure, modify, extend.

The ADT interface was chosen in order to faithfully reflect the intuitive and natural set notation. The final design is an extension of the Pascal model. It is based on infix operators instead of prefix functional notations of other languages and tools.
The set representation and the implementation of the operations, hidden and inaccessible to client programs, admits several options. Three classical representations can be indistinctly used: an array, a characteristic vector, and a linked list. The kind of problems and programs will select the proper alternative, ideally after take measurements.

The C++ language has features to efficiently write generic ADTs: classes, overloading, templates, and other minor helps. Additionally, it provides features to object orientation (inheritance, dynamic binding) that would permit to reuse the ADT code.

In sum, the implementation of sets, in/for the C++ language, represents an educational and practical combination that summarizes, in a versatile tool, the language potentiality.

References


[Alvarez93] J.Alvarez; C++: Attacks and Defenses; I Chilean Computing Congress; Santiago, Chile; October 1993

[Coplien92] J.O.Coplien; Advanced C++: Programming Styles and Idioms; Addison-Wesley; 1992 DIT-UPM; may 1993

[Ellis90] Ellis, M.; Stroustrup, B.; The Annotated C++ Reference Manual; Addison-Wesley; 1990


[Lea91] D.Lea; Users Guide to GNU C++ Library; Free Software Foundation Inc; February 1991

[McCacken87] D.McCracken, W.I.Salmon; A second course in Computer Science with MODULA-2; John Wiley; 1987

[Reilly89] E.D.Reilly, F.D.Federighi; Pascalgorithms; Houghton Mifflin Co.; 1989

[Stroustrup91] Stroustrup, B.; The C++ Programming Language; Addison-Wesley; 1991


[Wegner92] P.Wegner; Dimensions of Object-Oriented Modeling; Computer Vol.25 No.10; October 1992