A Critique of the Advanced Placement C++ Subset

Mordechai Ben-Ari
Department of Science Teaching
Weizmann Institute of Science
Rehovot 76100 Israel
ntbenari@wis.weizmann.ac.il

Kevlin Henney*
QA Training Ltd
Cecilly Hill Castle, Cirencester
Gloucestershire GL7 2EF
United Kingdom
kevlin_henney@qatraining.com

Abstract

The Educational Testing Service has decided that the Advanced Placement Examination in Computer Science will use the C++ programming language in place of Pascal. They have designed a subset of the language to be used in high school courses. This paper claims that the subset is deficient in two areas: (a) the subset is vague on which language features it contains, and (b) the proposed class library diverges significantly from the draft Standard Library.

These problems stem from an attempt to come to terms with non-complying implementations. We believe that the AP C++ subset should be defined on educational criteria alone; software and textbook authors and publishers would then adapt their course material to the language subset.

Introduction

The Educational Testing Service (ETS) has decided that the C++ language will be used on the Advanced Placement Computer Science (APCS) Examinations [2]. C++ will replace Pascal which is the current language of the examinations. The change is justified by the following considerations:

- The standard Pascal language has no facilities for encapsulation and abstraction which are an essential part of modern CS education, even at an introductory level.
- Pascal is not used extensively in industry, so new graduates may not be adequately prepared when they leave school. This point is especially important in community colleges and co-op programs, where students are expected to be immediately employable without the luxury of a full four years to explore alternative languages and systems.

C++ clearly fulfills these requirements for a first programming language.

The emphasis in modern CS1-2 courses is on algorithms, rather than on language syntax and semantics. Accordingly, the AP course in computer science has similar aims:

> Computer Science A emphasizes programming methodology and procedural abstraction. It also includes the study of algorithms, data structures, and data abstraction, . . . [2]

Typical questions on AP examinations ask the student to choose appropriate data structures, to trace executions of programs through procedure invocations (including recursive invocations), and to match specifications and programs [1]. Most of these questions are quite language-independent, and can be taught just as well in C or C++ as in Pascal. Thus the main impetus for the change is that C++ has constructs for encapsulation and abstraction (classes and operator overloading) that are lacking in C and Pascal. Most, if not all, CS educators believe that these concepts should be taught as early as possible.

Due to the complexity of C++ and to the plethora of dangerous features inherited from C, there has been extensive opposition in the CSE community to this decision. We promise not to continue the debate in this

*Member of the British Standards Institute C++ Standards Committee.

paper; instead we turn to an analysis of the subset of the language proposed by the ETS.

The AP C++ Subset

C++ is a very large language, and clearly there is no intention, nor is there any need, to teach the entire language in high school or in first-year college courses. Since the APCS examination is a standardized test, a standardized subset is necessary, and the ETS has set up a committee to define an appropriate subset.

But designing a language subset is a tricky business—almost as tricky as designing a language! For the APCS C++ subset, some decisions are clearly mandated by the intended use of the language: high-school students are not expected to learn about inheritance and dynamic polymorphism. Other decisions are not so clear-cut, and we wish to take issue with the proposed subset. Our objections fall into two categories:

- A language subset should be defined which is independent of any features of existing implementations. Such a subset would then guide textbook authors and software publishers in the preparation of course materials.
- The definition of the C++ language includes the definition of a standard library—the draft Standard Library (SL). The subset should include a subset of this library, rather than an ad-hoc library.

Define a Language, Not an Implementation

Exceptions In [4] we read:

> When all compilers support exception handling, this will be the preferred method for error handling despite the somewhat cumbersome syntax. Because not all compilers support exception handling, the AP C++ classes are written in two ways: ...

Exceptions are not just some optional feature, but a core construct of the C++ language. A subset designer must make a clear choice whether exceptions are part of the language or not. The error-handling mechanism significantly affects how one teaches algorithms. Possibilities are:

- Define a precondition for each program unit, and ignore the behavior on input values which falsify the precondition.
- Check input values and return an error code.
- Raise and handle exceptions.
- Use input assertions: preconditions that are executable and abort if false.

The choice of an error-handling mechanism for a specific project is a difficult software-engineering task, beyond the reach of high-school students.

In high-school CS material prepared by a team led by the first author, a conscious decision was made to ignore error handling, because Pascal lacks an appropriate construct. Instead, the first of the above options was chosen. If a decision is made to transition to C++, a decision must be made whether to continue the situation forced upon us by Pascal, or to modify the approach and use exceptions. The inclusion or exclusion of exceptions in the de-facto standard APCS subset would obviously be a major consideration in the decision.

Boolean type The C++ standard now includes the `bool` type. The keywords `bool`, `true` and `false` are reserved, and since `bool` is a full-fledged type, overloading is allowed:

```
void func(int value) { ...}
void func(bool value) { ...}
```

The APCS subset propose to allow a well-known kludge, defining `bool` as a synonym for `int`:

```
typedef int bool;
const int false = 0;
const int true = 1;
```

This causes two problems: (a) since `bool`, `true`, and `false` are identifiers rather than keywords, they can be hidden in inner scopes; (b) the overloaded definition of `func` with a `bool` parameter is considered to be an illegal duplicate definition. Thus the declarations in the subset will cause confusion for students and teachers.

Arithmetic precision The subset could specify a minimum 32-bit precision, in which case arithmetic precision need not be discussed. In any case, there is no need to use precision modifiers since single-word type specifiers (`short`, `long`) are standard ([3], §7.1.5.2).

Casts Old-style casts are not necessary, since one can use function casts even for long integers: `long(i)`. It is not clear from [4] whether the subset intends to teach and test implicit casts which are a central feature of C++, but since the subset includes overloading, a discussion of implicit casts and ambiguities becomes essential:

```
void func(double);
void func(char);
func(0);
```

The call is ambiguous because the literal `0` can be implicitly converted both to `double` and to `char`.

To summarize, the ETS is clearly worried about implementations that do not comply with the proposed standard. However, the standard will be finalized soon, and there is no reason to suppose that vendors will not support the full language in the near future. Furthermore, a clear subset language definition would encourage the development of educational development environments supporting it.

Subset the SL, Don't Rewrite It

Like other modern object-oriented languages the C++ language includes a comprehensive library, the SL. When the standard is finalized and implementations comply with it, one will not be able to claim to program in C++ without knowledge of the SL.

The library is extremely complex. We do not expect a high-school student (or teacher!) to understand the full declaration of the string concatenation operator ([3], §21.1):

```
template <class charT,
        class traits,class Allocator>
  basic_string<charT,traits,Allocator>
    operator+(const basic_string
      <charT,traits,Allocator>& lhs,
    const basic_string
      <charT,traits,Allocator>& rhs);
```

Instead, one should present the students with a subset implementation that is fully compatible with the following definition which is given in the SL:

```
typedef basic_string<char> string;
```

An SL-compatible interface is easy for a C++ programmer to understand:

```
string operator+(
    const string& lhs,
    const string& rhs);
```

and there is no need to rename the class to `apstring`. Certainly, there is no reason to change the names of functions of the SL; for example, `apqueue` uses `isEmpty` and `length`, where the SL uses `empty` and `size`!

Furthermore, there are areas where the APCS subset diverges completely from the SL:

Strings The `apstring` class is implemented in terms of null-terminated C strings, whereas C++ strings do not have this limitation. Also, the implementation does not fully conform to the standard specification:

```
apstring s = "hello", t = "world";
s[0] = '\0';
t = s;
```

Even though the strings `t` and `s` are "equal", `t[1]==s[1]` is false!

Containers Stack and queue classes in the SL are templates which are intended to be instantiated with an implementation class:[1]

```
template <class T,
  class Container=deque<T> > class queue;
```

According to the standard ([3] §23.2.4), appropriate implementation container classes for a queue are lists or deques; and for a stack they are vectors, lists or deques. We recommend that the subset implementation use lists rather than the default deques, but in any case, vectors (chosen for the APCS subset) are not considered an appropriate implementation for queues.

Furthermore, the vector class is only approved for implementing a stack because adding and removing from the end is an operation with amortized constant time cost—which is not the way that the APCS stack is implemented.[2]

Matrices The APCS subset includes a two-dimensional matrix class which is not included in the SL. The rationale is not explicitly stated, but we assume that the class is included to teach two-dimensional numeric algorithms. Unfortunately, the implementation of matrices is flawed: the matrix class is implemented as a vector of vectors, which is both redundant (since the data member `myCols` is defined) and incorrect (since "ragged" matrices can be created):

```
matrix<int> m(2,2);
```

[1]Note: the declaration in [3], §23.2, has been superseded by the one given here.

[2]The implementation of the stack reallocates the entire array when pushing onto a full stack.

```
vector<int> v(1);
matrix(0) = v;          // Ragged matrix !
matrix[0][1] = 0;       // Fails !
```

The implementation of efficient two-dimensional matrices should be directly in terms of two-dimensional arrays, range-checked using the additional data members `myRows` and `myCols`. The resize operation should have an explicit justification, and, if included, the treatment of size mismatches must be carefully specified.
In summary, we believe that:

- The APCS subset should include the list class.
- The class interfaces should be true subsets of the standard classes, and the implementations should be compatible with the intentions of the standard.
- Additional classes should be carefully implemented.

Additional Comments

Elementary types We do not agree with the decision to omit `enum` and `typedef`. C++ is being taught as a multiparadigm language, and leaving these out creates too large a gap between the idea of primitive types and full-blown type abstraction in the form of classes. Classes cannot really be taught until after expressions, functions and parameters; `enum`'s and `typedef`'s provide a useful, indeed essential, bridge to the idea of naming concepts.
Expressions The subset excludes the increment (`++`) and decrement (`--`) operators except in trivial cases. But the intent of the change to C++ is to teach students a widely used language. Whether we like it or not, the use of these operators in expressions is a well-known paradigm intensively used by C and C++ programmers. At the very least, students should should study hese constructs so that they can read existing software.

Conclusion

Defining a programming language, even a subset of an existing language, is very difficult. We believe that the AP C++ subset needs further work:

- The language defined should be a true subset, both of the C++ language and of its library.
- The subset should consider the long term: pulling technology to its educational agenda, rather than adapting itself to existing technology.
- The subset should be developed by a team that includes both CS educators and C++ “language lawyers” who are familiar with the ramifications of language decisions.

References

[1] Educational Testing Service. The 1992 Advanced Placement Examinations in Computer Science and Their Grading. 1993.

[2] Educational Testing Service. Advanced Placement Course Description: Computer Science. May 1996.

[3] Andrew Koenig. C++ Standard Draft Proposal. X3J16/95-0087, April 1995.

[4] Owen Ostrachan and Susan Horowitz. Second Call for Comments: C++ in APCS. http://www.cs.duke.edu/~ola/ap/cplus.html.