1 Introduction

In this short note we describe a simple subset smallCpp of the programming language C++ that can be used in first year introductory programming classes at universities. It is intended to be as simple as possible, while still being powerful enough to allow for decent programs. However, we impose several obligations, that are usually optional, e.g., regarding layout. We adhere to the following principles:

- All smallCpp programs can be compiled and run without errors and warnings by `g++`. Of course, this depends on versions and error levels, but we will in some sense be rather strict here.
- We use as few constructs as possible. We will not explain the different constructs in detail here.

Lines in smallCpp programs are at most 70 characters in length. Only characters with ASCII values from 0 up to and including 127 can be used. Only `//` comments can be used, so no `/* ... */` .

2 Types and variables

The only basic types that are allowed are `bool` (with only values `true` and `false`, no 0/1), `char`, `int` and `double`. Notice that for `int` and `double` we do not specify their number of bytes.
We can use single and double arrays. All array bounds must be specified by constants (by the way, no `#define`s), no numbers allowed. No possibility for `new` and `delete` here. In function headings the first `[ ]` must contain exactly one space.
Programmers can define `class`es in the usual way, even with the unfortunate `;` at the end. Only `private` and `public` members are allowed, and all must be specified as such.
Finally, we have pointer types, whose variables can only point to (address) objects of self-defined `class`es. We use `nullptr`, and not `NULL`. The `->` notation must be used to address members. (By exception, `*this` may be used to refer to the whole object itself, if really needed.)
All variables must be declared at the start of the function they are local in (or, of course, in the parameter list of their function); all declarations end with a comment. No global variables are allowed, only global constants.

3 Control structures

We will only use `if`, `while` and `for` statements. (And `switch` in some very special situations.) Also, `for` must be used only in a "simple" way, with all three components. A `do-while` statement is not allowed. Furthermore we have `new` and `delete` to create and destroy objects of self-defined `class`es. (No `delete [ ]`.)
Only `cin` and `cout` may be used to generate in/output from the keyboard and to the screen.
No usage of the ternary operator `...?...:...` is allowed. And no `,`-operator. The operators `++` and `--` are allowed, but only immediately following variables.

4 Layout

First of all, we expect layout to be internally consistent. Programmers use a fixed indentation, say $t = 2$ (or 3, or 4) spaces, and tabs are not allowed. For all statements curly braces `{ ... }` are used, even for single ones. If $k \geq 0$ "open" curly braces have passed, a line starts with exactly $k \times t$ spaces.
Every line contains at most one statement. Multiple declarations/definitions like `int a, b, c;` are not allowed for the moment: all variables need a line of their own, ending with a comment.
A starting `{` is on the same line as its statement/function, and ends that line — apart from optional comment. The concluding `}` is indented in the same way as the statement/function that started it, and is on an empty

line (perhaps with comment) by itself, implying that the line starts with $(k-1) \times t$ spaces. Alternatively, the opening `{` is allowed on a line by itself, indented in the same as the corresponding `}`.
Functions are separated by one or more empty lines, and have at least one comment line above their heading. Binary operators (`<<`, `>>`, `<`, `>`, `==`, `=`, `!=`, `+=`, `-=`, `<=`, `>=`, `+`, `-`, `*`, `/`, `%`, `&&`, `||`; no others are allowed) are always surrounded by at least one space on both sides. In particular this holds for the assigment operator `=`. It also holds for `&` in call by reference notation in function headings, the only place where `&` can be used. When using several `<<`'s, if on different lines, but belonging to the same `cout`, the first ones on a line start at the same indentation; they are not allowed at the end of a line. By the way, `>>` is allowed exactly once after `cin`.

5 Final details

The only keywords that were not mentioned so far, but that can be used, are `#include`, `void`, `main`, `const`, `else` and `return` (in particular: no `goto`, `do`, `static` (for the moment), `struct`, `scanf`, `printf`, `malloc`, ...). And of course those in `using namespace std;` .
When using *casting*, some flexibility is allowed.
We hope that SMALLCPP will be a fruitful addition to the world of programming.

Example program

```
1
2  // example.cc: a simple smallCpp program
3
4  #include <iostream>
5  #include <cstdlib>  // so rand ( ) will probably work
6  using namespace std;
7
8  const int MAX = 100; // for array length
9
10 // returns maximum of n-element array A
11 int maximum (int A[ ], int n) {
12   int i;           // counter
13   int maxi = A[0]; // for maximum value
14   for ( i = 1; i < n; i++ ) { // start from 1
15     if ( A[i] > maxi ) {
16       maxi = A[i];
17     }//if
18   }//for
19   return maxi;
20 }//maximum
21
22 // main
23 int main ( ) {
24   int B[MAX]; // the array
25   int i;      // counter
26   srand (42); // initialize random number generator
27   for ( i = 0; i < MAX; i++ ) { // fill the array
28     B[i] = rand ( ) % 10000;
29   }//for
30   cout << "Maximum value is " << maximum (B,MAX)
31        << endl;
32   return 0;
33 }//main
```