TECHNICAL REPORT

ISO/IEC TR 18015

First edition
2006-09-01

Information technology — Programming languages, their environments and system software interfaces — Technical Report on C++ Performance

Technologies de l'information — Langages de programmation, leurs environnements et interfaces du logiciel système — Rapport technique sur la performance C++

Reference number
ISO/IEC TR 18015:2006(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 18015, which is a Technical Report of type 3, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 22, *Programming languages, their environments and system software interfaces.*

Introduction

"Performance" has many aspects – execution speed, code size, data size, and memory footprint at run-time, or time and space consumed by the edit/compile/link process. It could even refer to the time necessary to find and fix code defects. Most people are primarily concerned with execution speed, although program footprint and memory usage can be critical for small embedded systems where the program is stored in ROM, or where ROM and RAM are combined on a single chip.

Efficiency has been a major design goal for C++ from the beginning, as has the principle of "zero overhead" for any feature that is not used in a program. It has been a guiding principle from the earliest days of C++ that "you don't pay for what you don't use".

Language features that are never used in a program should not have a cost in extra code size, memory size, or run-time. If there are places where C++ cannot guarantee zero overhead for unused features, this Technical Report will attempt to document them. It will also discuss ways in which compiler writers, library vendors, and programmers can minimize or eliminate performance penalties, and will discuss the trade-offs among different methods of implementation.

Programming for resource-constrained environments is another focus of this Technical Report Typically, programs that run into resource limits of some kind are either very large or very small. Very large programs, such as database servers, may run into limits of disk space or virtual memory. At the other extreme, an embedded application may be constrained to run in the ROM and RAM space provided by a single chip, perhaps a total of 64K of memory, or even smaller.

Apart from the issues of resource limits, some programs must interface with system hardware at a very low level. Historically the interfaces to hardware have been implemented as proprietary extensions to the compiler (often as macros). This has led to the situation that code has not been portable, even for programs written for a given environment, because each compiler for that environment has implemented different sets of extensions.

Participants

The following people contributed work to this Technical Report:

Dave Abrahams
Mike Ball
Walter Banks
Greg Colvin
Embedded C++ Technical Committee (Japan)
Hiroshi Fukutomi
Lois Goldthwaite
Yenjo Han
John Hauser
Seiji Hayashida
Howard Hinnant
Brendan Kehoe
Robert Klarer
Jan Kristofferson
Dietmar Kühl
Jens Maurer
Fusako Mitsuhashi
Hiroshi Monden
Nathan Myers
Masaya Obata
Martin O'Riordan
Tom Plum
Dan Saks
Martin Sebor
Bill Seymour
Bjarne Stroustrup
Detlef Vollmann
Willem Wakker

Information technology — Programming languages, their environments and system software interfaces — Technical Report on C++ Performance

1 Scope

The aim of this Technical Report is:

- to give the reader a model of time and space overheads implied by use of various C++ language and library features,
- to debunk widespread myths about performance problems,
- to present techniques for use of C++ in applications where performance matters, and
- to present techniques for implementing C++ Standard language and library facilities to yield efficient code.

As far as run-time and space performance are concerned, if you can afford to use C for an application, you can afford to use C++ in a style that uses C++'s facilities appropriately for that application.

This Technical Report first discusses areas where performance issues matter, such as various forms of embedded systems programming and high-performance numerical computation. After that, the main body of the Technical Report considers the basic cost of using language and library facilities, techniques for writing efficient code, and the special needs of embedded systems programming.

Performance implications of object-oriented programming are presented. This discussion rests on measurements of key language facilities supporting OOP, such as classes, class member functions, class hierarchies, virtual functions, multiple inheritance, and run-time type information (RTTI). It is demonstrated that, with the exception of RTTI, current C++ implementations can match hand-written low-level code for equivalent tasks. Similarly, the performance implications of generic programming using templates are discussed. Here, however, the emphasis is on techniques for effective use. Error handling using exceptions is discussed based on another set of measurements. Both time and space overheads are discussed. In addition, the predictability of performance of a given operation is considered.

The performance implications of IOStreams and Locales are examined in some detail and many generally useful techniques for time and space optimizations are discussed.

The special needs of embedded systems programming are presented, including ROMability and predictability. A separate chapter presents general C and C++ interfaces to the basic hardware facilities of embedded systems.

Additional research is continuing into techniques for producing efficient C++ libraries and programs. Please see the WG21 web site at www.open-std.org/jtc1/sc22/wg21 for example code from this Technical Report and pointers to other sites with relevant information.

2 Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 14882:2003, *Programming Languages – C++*.

Mentions of "the Standard" or "IS" followed by a clause or paragraph number refer to the above International Standard for C++. Section numbers not preceded by "IS" refer to locations within this Technical Report.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1

ABC

commonly used shorthand for an **A**bstract **B**ase **C**lass – a base class (often a virtual base class) which contains pure virtual member functions and thus cannot be instantiated (§IS-10.4)

3.2

access method

refers to the way a memory cell or an I/O device is connected to the processor system and the way in which it is addressed

3.3

addressing range

portion of the total of memory addresses accessible through processor instructions

NOTE A processor has one or more addressing ranges. Program memory, data memory and I/O devices may have special ranges which can only be addressed with special processor instructions. A processor's physical address and data bus may be shared among multiple addressing ranges.

3.4

address interleave

gaps in the addressing range which may occur when a device is connected to a processor data bus which has a bit width larger than the device data bus

3.5

cache

buffer of high-speed memory used to improve access times to medium-speed main memory or to low-speed storage devices

NOTE If an item is found in cache memory (a "cache hit"), access is faster than going to the underlying device. If an item is not found (a "cache miss"), then it must be fetched from the lower-speed device.

3.6

code bloat

generation of excessive amounts of code instructions, for instance, from unnecessary template instantiations

3.7

code size

portion of a program's memory image devoted to executable instructions

NOTE Sometimes immutable data also is placed with the code.

3.8

cross-cast

cast of an object from one base class subobject to another

NOTE This requires RTTI and the use of the `dynamic_cast<...>` operator.

3.9

data size

the portion of a program's memory image devoted to data with static storage duration

3.10

device

I/O Device

term used to mean either a discrete I/O chip or an I/O function block in a single chip processor system

NOTE The data bus bit width is significant in the access method used for the I/O device.

3.11

device bus

I/O device bus

data bus of a device

NOTE The bit width of the device bus may be less than the width of the processor data bus, in which case it may influence the way the device is addressed.

3.12

device register

I/O device register

single logical register in a device

NOTE A device may contain multiple registers located at different addresses.

3.13

device register buffer

multiple contiguous registers in a device

3.14

device register endianness

endianness for a logical register in a device

NOTE The device register endianness may be different from the endianness used by the compiler and processor.

3.15

down-cast

cast of an object from a base class subobject, to a more derived class subobject

NOTE Depending on the complexity of the object's type, this may require **RTTI** and the use of the `dynamic_cast<...>` operator.

3.16

Electrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory

EEPROM

similar to **flash memory** (sometimes called flash EEPROM), the principal difference is that EEPROM requires data to be erased and written one byte at a time whereas flash memory requires data to be erased in blocks and written one byte at a time

NOTE EEPROM retains its contents even when the power is turned off, but can be erased by exposing it to an electrical charge.

3.17

endianness

describes the layout in memory of the 0 and 1 bits which together represent a value

NOTE **Big-endian** and **little-endian** refer to whether the most significant byte or the least significant byte is located on the lowest (first) address. If the width of a data value is larger than the width of data bus of the device where the value is stored the data value must be located at multiple processor addresses.

3.18

embedded system

program which functions as part of a device

NOTE Often the software is burned into firmware instead of loaded from a storage device. It is usually a freestanding implementation rather than a hosted one with an operating system (§IS-1.4¶7).

3.19

flash memory

non-volatile memory device type which can be read like **ROM**

NOTE Flash memory can be updated by the processor system. Erasing and writing often require special handling. Flash memory is considered to be ROM in this document.

3.20

heap size

portion of a program's memory image devoted to data with dynamic storage duration, associated with objects created with `operator new`

3.21

interleave

see **address interleave**

3.22

Input/Output

I/O

term used in this TR for reading from and writing to **device register**s (see §8)

