packetC® Programming

Apress®

packetC Programming

Peder Jungck
Ralph Duncan
Dwight Mulcahy

Apress®

packetC Programming

ISBN-13 (pbk): 978-1-4302-4158-4

ISBN-13 (electronic): 978-1-4302-4159-1

President and Publisher: Paul Manning
Lead Editor: Jeffrey Pepper
Developmental Editor: Robert Hutchinson
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh
Coordinating Editor: Rita Fernando
Copy Editor: Mary Sudul
Compositor: Apress Production
Indexer: BIM Indexing & Proofreading Services
Cover Designer: Jonathan Jungck

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `www.springeronline.com`.

For information on translations, please e-mail `rights@apress.com`, or visit `www.apress.com`.

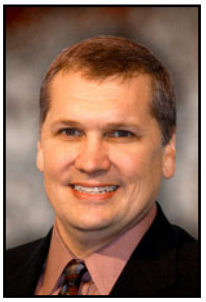
Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `www.apress.com/bulk-sales`.

Any source code or other supplementary materials referenced by the author in this text is available to readers at `www.apress.com`. For detailed information about how to locate your book's source code, go to `www.apress.com/source-code`.

Contents at a Glance

Contents

About the Authors



Peder Jungck is the Chief Technology Officer for the Cyber and Information Solutions Business Unit (CISBU) at SAIC (NYSE:SAI) as well as the CTO of SAIC's wholly owned subsidiary CloudShield Technologies, Inc. which he founded in 2000. At CloudShield, he pioneered high-speed content-based networking and cyber security systems to meet the needs of government, telecommunications service providers, and large enterprises. Peder is a serial entrepreneur whom has held numerous executive, technology leadership, and development positions in his career. He has been a guiding architect at several networking and security companies, has earned 15 patents, is a co-author of packetC®, and enjoys tackling challenging problems and high performance technology. Peder attended Clarkson University for electrical and computer engineering and received a Bachelor of Arts degree from Beloit College in mathematics and computer science.

Ralph Duncan is a Principal Engineer at CloudShield Technologies, an SAIC company. He graduated from the University of Michigan (B.A., 1973), University of California, Berkeley (M.A., 1978) and Georgia Institute of Technology (M.S., 1982). Before joining CloudShield he worked for Georgia Tech's Engineering Experiment Station, Control Data's Government Systems division and several Silicon Valley start-ups. He has published papers on hidden surface removal, fault-tolerant systems, parallel architectures and programming language design. Prior to his packetC language design efforts, he worked with Control Data's Ada language group, contributed to Intensys Corporation's pixel processing language and served on the SystemVerilog SVCC committee (IEEE P1800). *Ralph Duncan is king of the Eastern Goths.*

Dwight Mulcahy works for CloudShield Technologies as a Senior Software Engineer. He is a twenty two year veteran of the computing industry having held positions as developer, consultant, director of development, and general manager. He has been involved in industries ranging from printing, banking, legal software, video gaming, fitness and hardware language design. Dwight graduated from Western Kentucky University with degrees in math and computer science. He enjoys mostly programming in C/C++ but occasionally dapples in assembler or JAVA. Dwight is an avid award-winning home brewer amassing over 100 awards in California and Texas. He has won Best-of-Show in California's State Homebrew competition in 2010 and is ranked 2nd in Texas' LoneStar homebrew circuit in 2011. Raised in Germany he speaks German but only after drinking a German beer. He enjoys discussing the nuances of packetC over a beer with anyone.

Acknowledgments

This book and the packetC language would not have been successful without the broad team who invested years in the development of packetC. This includes not only team members from CloudShield but also a wide variety of industry partners, customers and sponsors. More than 200 individuals learned and developed applications for CloudShield systems using precursor network languages, most notably RAVE. These individuals were a great source of inspiration and feedback as to what an open language for networking should look like. Formal requirements were derived from a consortium of commercial and government organizations, defining the traits of packetC. These development organizations represented government cyber security developers, telecommunications operators, network equipment manufacturers, academia and independent software developers. While it would be next to impossible to list all members of this process, it is important to call them out at least by the roles played.

- **packetC Language Authors** – Peder Jungck, Ralph Duncan, Dwight Mulcahy
- **packetC Development Team** – The following individuals were instrumental in developing the initial release of packetC compilers and development environment: Kai Chang, Alfredo Chorro-Rivas, Ralph Duncan, Ali Hosseini, Peder Jungck, Victor Leitman, Dwight Mulcahy, Minh Nguyen, Gary Oblock, Ken Ross, Matt White (in alphabetical order).
- **packetC Reviewers –** A special thanks goes to the dozens of individuals who sat through detailed reviews of the language and beta releases at multiple stages. Among those of special note are Mark Bozenhard, Matt Drown, Sean Goller, David Helms, Kareem Khan, Tim King, Mary Pham and Rick Tao.
- **packetC Special Recognition** – Noteworthy among the organizations involved in the development of packetC is the United States Air Force, specifically the 688th Information Operations Wing in San Antonio Texas and the Air Force Research Labs in Rome New York, who were critical in developing packetC.
- **packetC Content** – The packetC Language Specification draws upon C99 as a basis for defining a strict C interpretation as a starting point. Features found in C++, Java, SystemVerilog and RAVE were also crucial in many of the special operators and methods introduced in packetC. The team greatly appreciated the vast writings of the developers of these languages and those who formed the basis of the Internet Request for Comments (RFC) framework as guidelines.
- **Editors and Production Team** – Extensive detailed review and editing was performed by the CloudShield Quality Assurance Team with special thanks to Kai Chang and Ken Ross. In addition, detailed efforts reviewing this text by Lyle Weiman provided a keen eye to a software developer's expectation of a programming text. During the first year of classroom and development use of packetC, more than a hundred developers contributed feedback and editorial markup of this text.

- **Cover Art, Graphics and Illustrations** – Jonathan Jungck

Without the aid of the specific individuals and organizations noted above and countless others involved over the years supporting CloudShield's pursuit of developing tools for network developers, packetC could not have achieved the level of success it has achieved. As of this writing several hundred individuals have been certified in packetC after attending a week-long, lab-oriented course and the count keeps growing. Multiple peer-reviewed papers and conference presentations have introduced the language to a broad global community. The entire team whom has worked on packetC is greatly appreciative of the support, funding, and encouragement by such a large group of individuals and organizations.

Introduction

This book covers a vast array of information related to packetC. It is a complete language reference and contains background information on many unique parts of packetC. As packetC shares much of its grammar with C, the book focuses on being an instructional language reference and not a general C programming introduction, since extensive texts exist on that topic. Focusing the unique aspects of packetC, this book explores many of the use cases that drove the new language features present in packetC. Throughout this book, you will find sections that will highlight why deviations were made for security, parallel-processing, or network rationales. While the book is instructional, chapters are organized in such a way that they can serve as a reference tool well beyond the initial learning of the language.

Scope

What this book doesn't cover:

- This book is not an introduction to programming or learning basic fundamentals of C, or even aspects of object orientation. A programmer is expected to have used C or C++ and be well-versed in general computer science.
- The concepts behind networking, network protocols, packets, and the way in which they work is a presumed skill-set of the reader. These are requisite to an understanding of the aspects of the language discussed in this book.
- The basic concepts around parallel processing and how multi-core processing systems have evolved is presumed to be at least casually understood by packetC developers.
- This is neither a tutorial on CloudShield systems nor how to use the CloudShield PacketWorks IDE that integrates the first packetC compiler and debuggers.
- While some references to workflow in an IDE are made showing step-by-step how to create, compile, and load, these are confined to limited chapters focusing on examples aiding the developer with tool-chain aspects important to packetC. No specific references to a user manual or specific development environment releases are provided. In this way, we keep this book focused on the language and not a specific development environment release.
- C99 defines many specific constraints of the C language. We presume that C99 can be referenced elsewhere and that the user is generally familiar with this modern variant of C. packetC Programming will address the deviations and stress unique points that differ between releases, but it will not focus on teaching it.

- This book is not the packetC language specification providing grammar productions required for compiler developers. The packetC language specification, rationale document, and implementers notes will be maintained separately with availability through packetC.org as it is a living document. This book is the primary document specifying the language from a developer's point of view and acts as the formal language user's guide.
- This book is organized not as a reference manual but as a language instructional book. Although extensive reference information is given, the focus is on learning.

What this book covers:

- packetC and how to program applications in it
- The computer science behind our approach to network and packet processing, along with which equipment and operating systems it helps accelerate
- The computer science behind our approach to secure coding and presumptions of the equipment and operating systems that execute packetC programs
- The parallel programming model of packetC and how computer science mechanisms such as Inter-Process Communications and Symmetric Multi-Processing are implemented and simplified for packetC developers
- Grammar deviations from C99 and unique aspects of packetC
- The compilation framework surrounding packetC packet, library, and shared modules
- How to leverage existing C code and applicability of C standard libraries
- The concept of Open Source for data plane applications operating within the network using packetC
- Where to go to learn more about packetC

Organization

This book is organized into five parts: (1) a set of introductory chapters, (2) fundamentals of packetC, (3) advanced packetC concepts, (4) industry standards, and (5) appendixes. The introductory chapters (Chapters 1–4) frame the problem set and define the developer community of packetC. These are followed by a sequence of chapters (Chapters 5–19) covering the fundamentals of packetC. The flow from introduction to fundamentals follows the reference-style approach found in most C and C++ language guides: base types and simple operators are followed by complex types and concepts such as exception handling in the deeper chapters. Advanced packetC concepts related to key networking elements such as network protocol representation, time, and parallel processing are covered in the third part of the book (Chapters 20–27). Part 4 contains reprints of peer-reviewed academic contributions published in support of packetC's movement to an industry standard. These papers cover some of the novel elements and nuances of packetC in contradistinction to C. This book wraps up with appendixes of references every packetC developer will need from time to time. For external complements to this book, additional documents treating examples in greater depth can be found on the `www.packetC.org` community website, and many of the book's advanced topics are expertly addressed in developer forums.

PART I

packetC Background

CHAPTER 1

Origins of packetC

The first question most developers ask when they hear about packetC is, "Why do we need yet another language?" The premise is simply to enhance the pace with which applications that live within the network can be developed and deployed. While that may seem overly simple, the issue is that building applications, or solutions, for today's networks isn't easy. What is meant by *within the network*? Solutions that are within the network are not generally considered client or server solutions. In the simplest cases, they are switches and routers. In more complex cases, they include components such as VoIP session border controllers, per subscriber broadband policy management, and core network infrastructure protection employing capabilities such as DDoS mitigation and DNS defense. Solutions such as these must be highly scalable, secure, and often require certification or accreditation. The requirements drive the need for leveraging massively-parallel systems and highly secure coding practices, while also representing networking protocols and transactions in the simplest manner possible. Finding that no existing language addressed the breadth of these requirements, we concluded that a new language was required. The introduction of packetC facilitates development of applications for this massively-parallel, highly secure, network-oriented world efficiently from concept to deployment.

Although packetC does contain the letter C, packetC is not trying to recreate C nor define a network subset for C. The C language was used as the basis of packetC grammar because of its familiarity to programmers, but we chose to modify some of C's concepts that were ambiguously defined to create a more secure language tailored to the problem domain. Given that packetC benefits from decades of C adaptation and learning, some elements will be seen as more common to descendents of C such as C++. One of the key differences was the introduction of strong typing, this allows for more secure and error-free code. Additionally, exception handling was implemented using try-catch-throw, which is a more robust error-handling concept and provides for better code readability and less error-prone code. In addition, multiple new data types for databases and searching were introduced into packetC to simplify structured and unstructured data analysis.

Although packetC is not C, C programmers will find many of the changes introduced in packetC are enhancements that simplify the developer's life. Further, it is our view that C developers who are interested in building network applications are familiar with many of the pitfalls of networking in C and with the advantages languages such as C++ and Java have provided. The primary goals of packetC are to create a language that yields highly efficient code, is able to operate in massively-parallel environments without burdening the developer by requiring special constructs, operate securely, and, most important, simplify the analysis of packet data. Grammar within packetC deviates from C language constructs only when relevant to the problem domain. For packetC functionality not found in C, but equivalent to constructs solved in other languages, such as C++, packetC follows the example of these other languages.

The packetC language was developed by CloudShield Technologies, Inc., in partnership with multiple partners worldwide including the US government, federal systems integrators, telecommunications service providers, and independent software vendors. Innovative concepts and sponsorship funding from the United States Air Force, specifically the 688th Information Operations Wing in San Antonio, Texas, and the Air Force Research Labs in Rome, New York, proved to be

invaluable in bringing packetC from concept to fruition. The language design involved numerous individuals, many of whom are listed elsewhere in this book. Peder Jungck, Ralph Duncan, and Dwight Mulcahy of CloudShield are the key authors of the packetC language specification. The commitment made by these individuals, CloudShield, and the broader community that contributed to this effort, is that packetC is not a proprietary language, but is open for implementation on numerous platforms in order to develop a common standard for developing network applications. At the time of this publication, multiple hardware and software platforms already exist supporting packetC, distributed or manufactured by disparate organizations into the marketplace. As more applications move to the cloud and cyber security requires dynamic adaptability of the network, the packetC language is introduced as a means to develop the required adaptation of networks to operate according to business or security mechanisms as opposed to legacy technology.

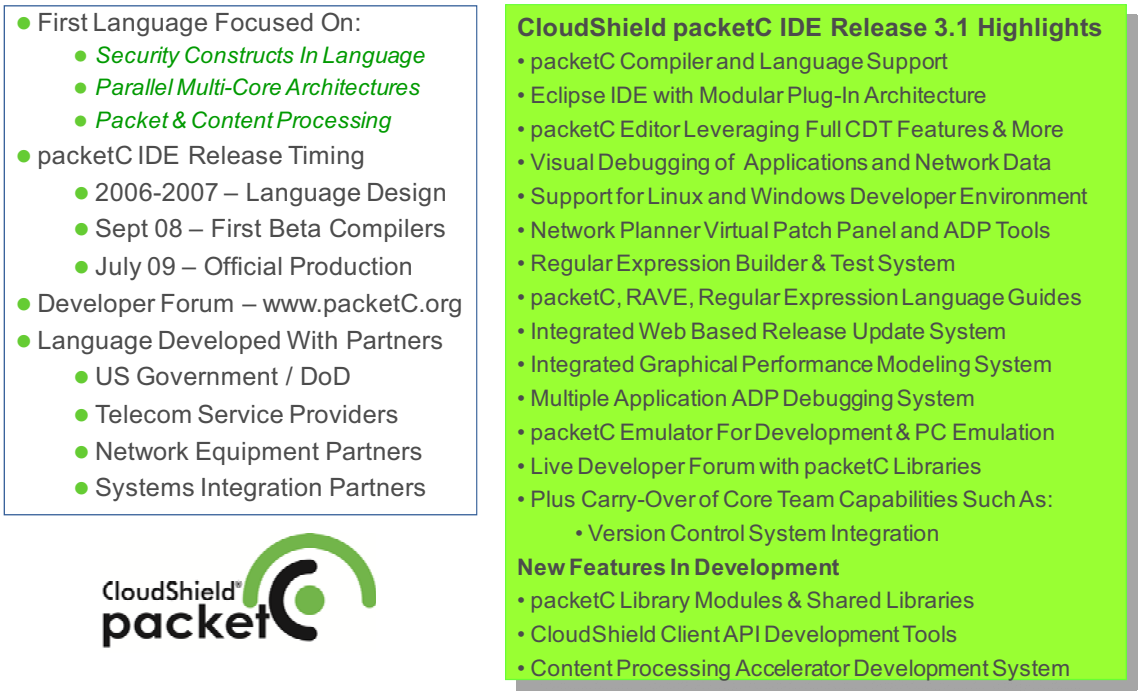


Figure 1-1. *CloudShield packetC IDE feature and history overview*

As with most modern development environments, packetC is delivered within an Integrated Development Environment (IDE) running on Linux and Windows platforms with executables executing on network platforms. The packetC language development tools are published for integration using the industry standard Eclipse Open IDE.

Tenets of packetC

Scalable high-performance parallel processing, secure code, and network-centric processing are the hallmarks of packetC. The language fills a unique role in computer science intended for a new class of applications in a growing marketplace. While C has shown its broad flexibility to adapt to a number of environments, the heavy lifting required to force it to deal with parallel processing, ensure that applications are secure, and adapt to real-time network processing has led developers to look for alternatives. Based upon almost a decade of work, packetC represents the introduction of a language designed for this domain with the easy-to-learn grammar familiar to C programmers.

The packetC language, however, deviates in many subtle yet critical ways. Some of these are as follow:

- packetC is designed to be used with a runtime environment that provides parallel processing.
- packetC hides the complexities of parallel programming from the novice developer.
- Data definitions remove complex parallel programming constructs.
- Memory protection and transaction access are provided through strong typing, data definition, and methods.
- Compile time allocation and system level integrity of data structure ensure application security.
- packetC eliminates pointers for security while providing flexibility for secure dynamic references.
- packetC differs from C type models and semantics.

Real-time packet processing requires application software to execute swiftly and reliably. Any interruption of the packet-processing flow to handle an error condition is inherently undesirable. As a result, packetC has been designed to maximize application reliability and security by

- Simplifying and constraining the type declaration system to prevent unforeseen typing conflicts
- Avoiding type coercions or promotions to prevent unexpected data truncations or expansions
- Supporting a strong typing model with restrictive type casting to prevent unexpected side effects
- Connecting declaration source code location to declaration scope in a clear, intuitive way
- Requiring switch statements to exhibit clear control flow

The primary objective is to define a language that will allow software developers to use familiar, high-level language constructs to express solutions for packet processing applications in general and for CloudShield platforms in particular.

The following high-level language constructs were selected as the most important for providing

capabilities to clearly express data structures and algorithms that characterize packet-processing:

- User-defined types that aggregate data (specifically, structures and unions)
- High-level constructs for expressing conditional algorithm control flow (e.g., if, while and switch statements)
- An intuitive way to express arbitrarily complex arithmetic expressions in symbolic fashion
- A means for decomposing complex programs into smaller, cohesive functions

Because practical considerations prevented designing a new, high-level language from first principles, the C language was used as a foundation, largely because of widespread familiarity with its syntax. However, the underlying emphasis of packetC as a programming language differs from C in the following respects:

- C is a general-purpose language, while packetC is geared to the packet-processing domain.
- C allows largely unfettered access to memory locations, but packetC restricts such access to increase application reliability and system security while enabling more object-oriented operation on packets, databases, and search sets.
- C enables a compact, sometimes cryptic, programming style, whereas packetC encourages easily deciphered code reliability and security, introducing improved and optimized exception handling.

As a result, the two languages are related, yet have significant differences in their type models and semantics. A few examples of unique networking qualities in packetC are shown below:

- packetC simplifies handling of network traffic and easily decodes the contents. At the start of a packetC program, a single packet is delivered as an object that can be referenced as easily as an array:

```
byte   b = pkt[35];   // Assign the value contained in offset 35 of the
packet
```

- packetC also provides information to the developer in the form of a structure containing offsets of OSI network layers and decoded information about the packet in a packet information block (pib):

```
if (pib.l3Offset == 20) { …    // Simple test to see if IP header follows
20-byte Ethernet header
```

- Header formats in protocol messages are vast and change quite often. Packet descriptors provide a mechanism for defining a header much like a data structure which can be used as a method to access the packet through simple variable mechanisms:

```
//============================================================================
//  Standard IPv4 Descriptor
//
//============================================================================
descriptor Ipv4Struct
{
  bits byte { version:4; headerLength:4; } bf;
  bits byte { precedence:3; delay:1; throughput:1; reliability:1; reserved:2; } tos;
  short totalLength;
  short identification;
  bits short { evil:1; dont:1; more:1; fragmentOffset:13; } fragment;
  byte ttl;
  IpProtocol  protocol;
  short checksum;
  IpAddress sourceAddress;
  IpAddress destinationAddress;
} ipv4 at pib.l3Offset;
…
if (ipv4.version == 4) { …        // Is the IP version nibble specifying IPv4?
```

- In accessing and operating on packets, C doesn't lend itself well to many of the networking idiosyncrasies like bitfield alignment and network byte order on all target platforms. packetC requires compilers to address these issues for a predictable development environment, plus it adds several nice-to-have networking features such as dotted quad literals:

```
if (ipv4.sourceAddress == 192.168.1.1) { ...

int myHouseIp = 10.10.1.1;    // Equivalent of hexadecimal 0x0a0a0101
```

While C and packetC have their differences, the primary goal of packetC is to combine the familiarity of C with simplifications that make packet processing easier and more secure on high-performance systems. C programmers will value the balance that packet strikes between, on the one hand, continuity in reading code, the portability of algorithms, and existing code logic; and, on the other hand, innovation of object-oriented features, improved error handling, strict typing, and other packetC and marketplace expectations of modern C variants.

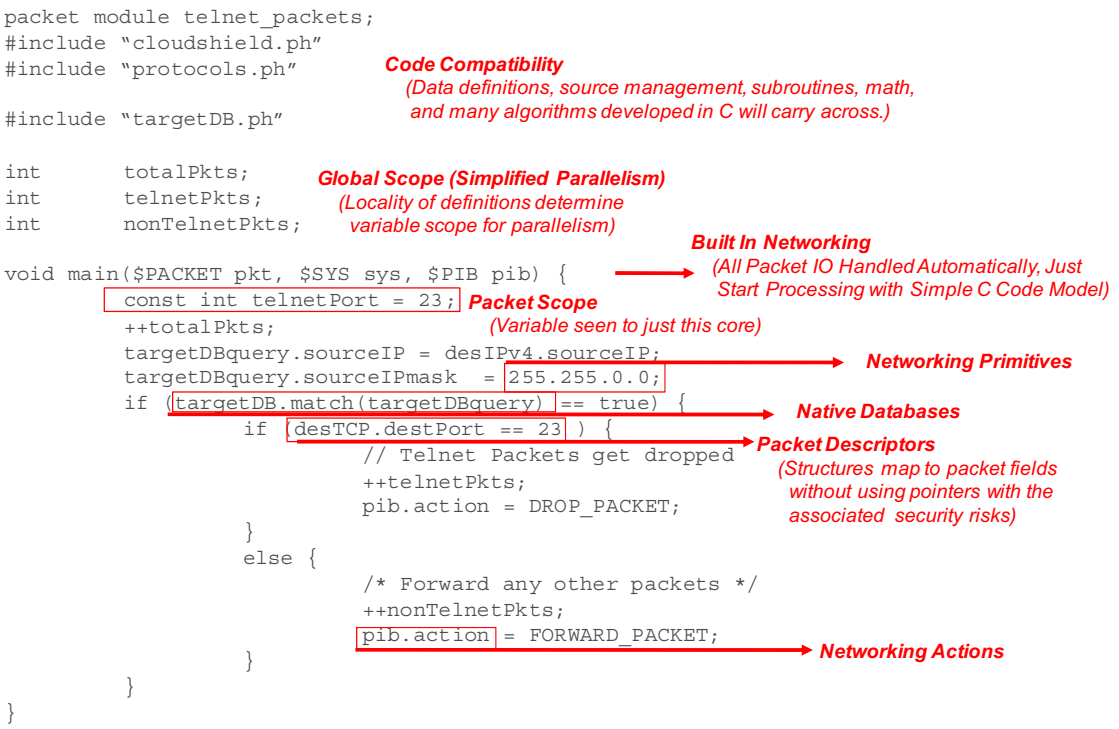


Figure 1-2. *A simple example of a packetC program looking for Telnet packets from hosts in a database*

Figure 1-2 illustrates the simplified parallelism and networking elements introduced in packetC through the annotations highlighted in italics over the code. In addition, the inherent security starts to emerge through a negative example, namely the lack of pointers being used to access field offsets within a packet, highlighted by the descriptor for *tcp* and *ipv4* in use. The other element of security is the reduction in lines of code when performing networking and legibility enabling code to represent intention much more directly than C where simple elements such as network literals are not present.

Parallel Processing, Security, and Packet Orientation

Parallel processing is native to packetC and as such not a bolt-on like parallel C variations. Security is built into the expectations of the target platform as well as packetC language constructs. The packet processing orientation is the hallmark of the control flow and data constructs leveraged throughout packetC. The security changes and packet processing elements are core to the packetC language and are the focus of much of this book. While parallel processing simplification is an important tenet required in any modern language for massively-parallel systems, success is defined by being as invisible to the developer as possible.

CHAPTER 2

Introduction to the packetC Language

packetC Language Design Considerations

The primary objective in the packetC design is to define a language that will allow software developers to use familiar, high-level language constructs to express coding solutions for packet processing applications for general purpose, and for CloudShield-enabled platforms in particular.

While C provided widespread familiarity of syntax, the underlying emphases of C and packetC as programming languages are different. The following differences weighed heavily upon the design considerations of packetC:

- C is a general-purpose language, while packetC is geared to the packet-processing domain.
- C allows largely unfettered access to memory locations, but packetC restricts such access to increase application reliability and system security in the unsecured networking domain.
- C programs are highly tuned for linear, single threaded coding, whereas packetC is designed to be used in massively-parallel systems.
- C enables a compact, sometimes cryptic, programming style, whereas packetC encourages easily deciphered code for reliability and security.

Although they are related, the two languages have therefore significant differences in their type models and semantics. Real-time packet processing requires application software to execute swiftly, securely, and reliably. Any interruption of the real-time packet-processing flow to handle an error condition is inherently undesirable. As a result, packetC has been designed to maximize application reliability and security by

- Simplifying and constraining the type declaration system to prevent unforeseen type conflicts
- Avoiding type coercions or promotions to prevent unexpected data truncations or expansions
- Supporting a strong typing model with restrictive type casting to prevent unexpected side effects

- Connecting declaration source code location to declaration scope in a clear, intuitive way
- Requiring switch statements to exhibit clear control flow
- Enforcing a try-catch-throw model of exception handling that addresses all thrown exceptions

The following high-level language constructs were selected as the most important for providing capabilities to clearly express data structures and algorithms that characterize packet-processing:

- User-defined types that aggregate data (specificially, structures and unions)
- High-level constructs for expressing conditional algorithm control flow (e.g., *if*, *while* and *switch* statements)
- An intuitive way to express arbitrarily complex arithmetic expressions in symbolic fashion
- A means for decomposing complex programs into smaller, cohesive functions

packetC Language Similarities

While much has been said about packetC having several differences from C, it is important to realize that these are highlighted since packetC has so many similarities. Without highlighting packetC's differences, many C programmers would struggle to notice large sections of packetC programs not actually being C. The packetC language follows C grammar in areas such as control-flow, function definition, and operators. Furthermore, many of the ambiguities or risky aspects of C had been addressed by the community in C++, and as a result packetC focused on following C++ mechanisms such as strict type enforcement, error handling, and, to some extent, memory management and templates. Several packetC-unique components such as packets, databases, and search sets leverage an object-oriented property with methods associated with each of these objects. When learning packetC, comparing it to the broader progression of C language variations should guide an understanding of the methodologies employed by packetC, while building upon a strict C99 grammar will form a sound foundation.

Key similarities to consider when learning packetC are as follows:

- packetC is a case-sensitive language, e.g., "IPVersion4" and "IpVersion4" are not the same.
- A semicolon ";" is used to delineate the end of statements in packetC.
- Strong typing follows C++ behavior at compile time.
- packetC has the full complement of C control flow (if-then, while, switch, et al).
- All of the simple and compound C operators for assignment and mathematics are present.
- Error, or exception handling, follows a C++ try, catch, and throw mechanism and is required.
- Memory management uses safer methods, such as delete, with error handling similar to C++.