DOWNLOAD PDF REDUCING COBOL COMPLEXITY THROUGH STRUCTURED PROGRAMMING

Chapter 1 : Structured Data Manager | Micro Focus

COSC or equivalent Structured programming principles and techniques, as implemented through the ANSI COBOL language; program design using top-down techniques; program and project documentation; introduction to sequential and random file algorithms and integrated file systems. Course Objectives Upon successful completion of this course, the student should be able to: Develop data definitions and record structures from user problem specifications. Distinguish the environmental differences between various computing platforms as they relate to the execution of COBOL programs. Recognize how COBOL employs structured programming techniques to solve programming oriented problems. Detailed Course Outline 1. Introduction to Structured Programming. Fixed-length versus variable-length records. COBOL data storage types and internal data representation. Simple conditional operations using the IF statement. Distinguishing between numeric and numeric-edited fields. Explain the difference between implied and actual decimal points. Learn the rules for signed numbers and editing characters. Predict the effect of data movement between fields. A study of the basic reference modification principles. Distinguishing between fixed and variable length records. Differentiating between a subscript and an index. Distinguishing between sequential table lookup, binary table lookup, and direct access to table entries. Implementation of a multidimensional table. Searching a multidimensional table. Differentiating between ascending and descending sorts; between major, intermediate, and minor keys. Defining a collating sequence. Distinguishing between a sort and a merge. A look at the general guidelines for report design. Defining control break; distinguishing between a single control break and multilevel control break programs. Distinguishing between running and rolling totals. Study of the single-level and two-level control break algorithms. Generalization to three-level, etc. Distinguishing between a called program and a calling program. Description of the purpose of the linkage-editor. Explanation of the meaning of unresolved external reference. Explanation of the three transaction types associated with file maintenance. The basic sequential update algorithm. Introduction to indexed and virtual sequential file implementations. Distinguishing between indexed and virtual sequential file organizations. Programming specifications for interactive file maintenance. Application of ergonomics in interactive program design. These will be based on material discussed in class. The examinations consist of short answer, explanation, analysis, and what-if questions. About six to seven projects of varying complexity each based on topics discussed so far in the semester. The standard grading scale will be used. Project comprises of a simple program furnished by the professor preferably with introduced errors. Students are informed of the errors and they employ the Language Sensitive Editor to eliminate the errors. Students then compile, link and execute the program for submission. Project should stress the use of the AT END scheme for detecting the end of file in program execution. Project comprises of simple data manipulation, preferably just moving input fields to output fields. Professor may introduce simple arithmetic with the project. Project comprises of a COBOL program that does arithmetic, preferable with some decision structures involved. Professor supplies the input file and the program specifications. The specification should include the requirement for a formatted output. Students will have to code the entire program. Professor supplies the program specifications and students code the program. Project would require students to code a multi-level control break program preferable a two-level control break. Two weeks A Sequential File Update program. Project would require students to update a master file with transactions either from a file or provided for interactive update. Preferably the specifications would require that student programs be able to handle multiple transactions belonging to particular records. Two weeks A Nonsequential File Update program. Project would require students to develop an interactive program that would randomly update a master file with transactions. From Micro to Mainframe. Fourth Edition, Prentice-Hall Inc.

Chapter 2 : US - IBM COBOL Structuring Facility V2

The scale is not strict and there is some overlap 1 and 2 are particularly good and 6 and 7 are particularly bad. The measures were developed by Glenford Myers "based on studying modules in each category and their relative effects on module reusability, error-proneness, independence and program maintainability and extensibility. In any design the aim would be to maximise the strength and realise the trade-off when this cannot be done. Coincidental Strength Either its function cannot be defined the only way to describe its function is to describe its logic i. Step down through what the code does from section to section OR It performs multiple completely unrelated functions. They are obviously a bad idea but are developed because of bad partitioning of existing programs and attempts at efficiency by storing duplicate code in a module. They are not independent and tend to be closely related to calling and called modules. Their interfaces are normally large and the chances of reuse are small. They degrade program maintainability and extensibility and require rewriting and restructuring. Logical Strength Performs a set of related functions, only ONE of which is explicitly selected by the calling module using an argument. There is a single interface for multiple functions. More cumbersome to use and reuse and harder to change. There is also a temptation for internal code to be "shared" between functions adding to maintenance problems. The motivation of locating for example all functions that relate to single data structure in the one place is good but it can be better achieved with an informational strength module. Classical Strength Also called Temporal strength. There are multiple sequential functions with a weak but non-zero relationship between them e. They are mainly related because they are carried out at the same time in execution terms. But they are related to many different external modules and these implicit relationships are hard to distinguish. Where should initialisation occur? Just prior to use. Procedural Strength Higher on scale than Classical and the boundary between easy and hard to maintain modules. Multiple functions with a sequential relationship between all the functions are implied by the problem statement: The order of execution is more important than with Classical. Problems arise because functions tend to be coded together and have interdependencies and there are still multiple functions. The elements of the module all share a data relationship but not a sequence relationship. For example they all share access to the same data structure. In each case all functions are carried out each time the module is called. Problems occur when not all functions are required. This makes the module hard to reuse and maintain. Plus there is the problem of sharing code between functions. Here the module has both a data and a sequence relationship such that data is passed in an assembly-line approach as output of one function step to input of the next. There is obviously a very close connection between the items. It is generally easy to maintain but does not complete one full function by achieving all its parts so it is not so reusable. This is the highest form of internal module strength. They carry out a single specific function. They are understandable from their name and argument list and they have excellent "black-box" features. Informational Strength Functional strength modules may often be externally inter-related coupled due to referencing the same data structures. Placing all such related functional strength modules within an overall module that hides the data structure creates an informational strength module. It contains multiple entry points for calling each individual function separately. All of the functions are related by a concept, a data structure or a resource that is hidden within the module. There are no control-flow connections between the logic of the separate functions. Although they can of course call each other. It is the essence of object-oriented design as this data plus functions is an object. It is viewed as slightly lower than functional because of the complexity of having several functions and the danger of coding them together, Object -oriented enthusiasts would maintain that it is a more appropriate design approach for larger systems. Write a sentence describing the purpose of the module and examine the sentence structure. Summed up with precise verb-object phrase.

Chapter 3 : Carma L. McClure | Open Library

Note: Citations are based on reference standards. However, formatting rules can vary widely between applications and fields of interest or study. The specific requirements or preferences of your reviewing publisher, classroom teacher, institution or organization should be applied.

The IBM extensions are summarised in the Implementation sub-section for the compiler later. Although there are some extensions common to these compilers the lack of a current standard means that compatibility is not guaranteed. A subset of the GY [16] document was offered to the joint effort by IBM and became the base document for standardization. The major features omitted from the base document were multitasking and the attributes for program optimization e. Proposals to change the base document were voted upon by both committees. In the event that the committees disagreed, the chairs, initially Michael Marcotty of General Motors and C. Hoare representing ICL had to resolve the disagreement. Further development of the language occurred in the standards bodies, with continuing improvements in structured programming and internal consistency, and with the omission of the more obscure or contentious features. Discussion of a single item might appear in multiple places which might or might not agree. It was difficult to determine if there were omissions as well as inconsistencies. Andrews of IBM undertook to rewrite the entire document, each producing one or more complete chapters. It was the first, and possibly the only, programming language standard to be written as a semi-formal definition. To fit a large compiler into the 44 kilobytes of memory available on a kilobyte machine, the compiler consists of a control phase and a large number of compiler phases approaching The phases are brought into memory from disk, one at a time, to handle particular language features and aspects of compilation. Each phase makes a single pass over the partially-compiled program, usually held in memory. Reflecting the underlying operating system, it lacks dynamic storage allocation and the controlled storage class. Unlike the F compiler, it has to perform compile time evaluation of constant expressions using the run-time library, reducing the maximum memory for a compiler phase to 28 kilobytes. Macros were defined to automate common compiler services and to shield the compiler writers from the task of managing real-mode storage, allowing the compiler to be moved easily to other memory models. The gamut of program optimization techniques developed for the contemporary IBM Fortran H compiler were deployed: This compiler went through many versions covering all mainframe operating systems including the operating systems of the Japanese PCMs. The team was led by Brian Marks. This format is interpreted by the Checkout compiler at run-time, detecting virtually all types of errors. Pointers are represented in 16 bytes, containing the target address and a description of the referenced item, thus permitting "bad" pointer use to be diagnosed. In a conversational environment when an error is detected, control is passed to the user who can inspect any variables, introduce debugging statements and edit the source program. Over time the debugging capability of mainframe programming environments developed most of the functions offered by this compiler and it was withdrawn in the s? UniPrise Systems , Inc. Additional data types and attributes corresponding to common PC data types e. Improvements in readability of programs â€” often rendering implied usages explicit e. The ordinal facilities are like those in Pascal , e. Competitiveness on PC and with C[edit] These attributes were added: The DATE pattern attribute for controlling date representations and additions to bring time and date to best current practice. Compound assignment operators a la C e. Additional parameter descriptors and attributes were added for omitted arguments and variable length argument lists. Program readability â€” making intentions explicit[edit] The VALUE attribute declares an identifier as a constant derived from a specific literal value or restricted expression. The package construct consisting of a set of procedures and declarations for use as a unit. It was heavily used by Daisy Systems for electronic design automation software on the "Logician" family of special-purpose workstations. It has been widely used in business data processing [49] and for system use for writing operating systems on certain platforms. The pioneering online airline reservation system Sabre was originally written for the IBM in

assembler. It remained a minority but significant player. First, the nature of the mainframe software environment changed. On mainframes there were substantial business issues at stake too. Compiler development was expensive, and the IBM compiler groups had an in-built competitive advantage. Many IBM users wished to avoid being locked into proprietary solutions. But a number of features of significance in the early implementations were not in the Standard; and some were offered by non-IBM compilers. And the de facto language continued to grow after the standard, ultimately driven by developments on the Personal Computer. IBM has continued to add preprocessor features to its compilers. The preprocessor treats the written source program as a sequence of tokens, copying them to an output source file or acting on them. Tokens are added to the output stream if they do not require action e. Subsequent occurrences of PI would be replaced by 3. The structure statements are:

Chapter 4 : Reducing COBOL complexity through structured programming - Carma L. McClure - Google Bo

Books by Carma L. McClure, Reducing COBOL complexity through structured programming, Case is Software Automation, The three Rs of software automation, Managing software development and maintenance, Software reuse techniques, Software reuse.

Discuss July Background[edit] In the late s, computer users and manufacturers were becoming concerned about the rising cost of programming. At a time when new programming languages were proliferating at an ever-increasing rate, the same survey suggested that if a common business-oriented language were used, conversion would be far cheaper and faster. Hawes , a computer scientist at Burroughs Corporation , called a meeting of representatives from academia, computer users, and manufacturers at the University of Pennsylvania to organize a formal meeting on common business languages. The delegation impressed Charles A. Portable programs would save time, reduce costs and ease modernization. It was attended by 41 people and was chaired by Phillips. They agreed unanimously that more people should be able to program and that the new language should not be restricted by the limitations of contemporary technology. A majority agreed that the language should make maximal use of English, be capable of change, be machine-independent and be easy to use, even at the expense of power. The short-range committee was given to September three months to produce specifications for an interim language, which would then be improved upon by the other committees. Work began by investigating data description, statements, existing applications and user experiences. While some members thought the language had too many compromises and was the result of design by committee , others felt it was better than the three languages examined. Some felt the language was too complex; others, too simple. Such features included boolean expressions , formulas and table subscripts indices. They fell short of expectations: Joseph Wegstein noted that "it contains rough spots and requires some additions", and Bob Bemer later described them as a "hodgepodge". The subcommittee was given until December to improve it. Despite being technically superior, FACT had not been created with portability in mind or through manufacturer and user consensus. We shortened it and got rid of a lot of unnecessary notation. Sammet of Sylvania Electric Products. The sub-committee did most of the work creating the specification, leaving the short-range committee to review and modify their work before producing the finished specification. COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith. The authors and copyright holders of the copyrighted material used herein are as follows: They have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications. A US Navy evaluation found compilation speeds of 3â€”11 statements per minute. By mid, they had increased to 11â€” statements per minute. It was observed that increasing memory would drastically increase speed and that compilation costs varied wildly: This was then replaced by the COBOL Extended specifications in , which introduced the sort and report writer facilities. They described new versions in , , and , including changes such as new inter-program communication, debugging and file merging facilities as well as improved string-handling and library inclusion features. It was also poor, lacking the funds to make public documents, such as minutes of meetings and change proposals, freely available. These made up 44 changes, which rendered existing statements incompatible with the new standard. The proposed standard commonly called COBOL differed significantly from the previous one, causing concerns about incompatibility and conversion costs. In January , Joseph T. Brophy described previous conversions of their million-line code base as "non-productive" and a "complete waste of our programmer resources". Fewer than a dozen of the responses were in favor of the proposed standard. In the same year, a National Bureau of Standards study concluded that the proposed standard would present few problems. Sixty features were changed or deprecated and many[quantify] were added, such as:

DOWNLOAD PDF REDUCING COBOL COMPLEXITY THROUGH STRUCTURED PROGRAMMING

Chapter 5 : PL/I - Wikipedia

MeasuringÂ· program complexity in a COBOL environment by JEAN ZOLNOWSKI and DICK B. SIMMONS Texas A&M Unil'ersity College Station, Texas INTRODUCTION Websterl defines complexity as the "quality or state of.

Chapter 6 : COBOL - Wikipedia

A frequently stated objective of structured programming is to control program complexity. However, since the notion of complexity is not well under-stood and the existing techniques for measuring complexity are crude, it is difficult to determine if indeed structured programming can achieve this objective.

Chapter 7 : cobol presentation

Measuring programmer productivity and estimating programming time and costs are among the most worrisome and persistent problems facing the programming manager.

Chapter 8 : Three Universal methods of reducing complexity

found: Her Reducing COBOL complexity through structured programming, , c t.p. (Carma L. McClure) CIP galley (Ph. D.; developed systems for SAMI, a subsidiary.