REGIONE AUTONOMA
FRIULI VENEZIA GIULIA

Concorso pubblico per esami per la copertura di n. 10 posti di categoria D, posizione economica D1, profilo professionale specialista tecnico, indirizzo informatico, con contratto di lavoro a tempo pieno e indeterminato, presso la Regione autonoma Friuli Venezia Giulia, anche per le esigenze dell'Organismo Pagatore Regionale (OPR FVG)

QUESITI PROVA ORALE

- Il procedimento amministrativo: l'avvio
- Il procedimento amministrativo: la conclusione
- Il responsabile del procedimento
- Il responsabile dell'istruttoria
- Il provvedimento amministrativo
- La motivazione del provvedimento amministrativo
- L'accesso agli atti della pubblica amministrazione
- Accesso generalizzato
- Accesso generico
- Il conflitto di interessi
- La trasparenza della Pubblica amministrazione
- Reati che possono essere commessi dai pubblici ufficiali
- Reati contro la pubblica amministrazione
- Il provvedimento autorizzativo
- Il provvedimento contributivo
- L'autotutela della pubblica amministrazione
- I provvedimenti legislativi
- La giunta regionale
- Il consiglio regionale
- Il Presidente della Regione
- Il Direttore generale
- Il codice di condotta e comportamento
- Gli atti di programmazione della Regione

- Il Piano della performance
- La pubblicazione di leggi e regolamenti regionali
- La pubblicazione degli atti e dei provvedimenti
- La pubblicazione nell'ambito della pubblica amministrazione
- Prevenzione della corruzione nella Pubblica Amministrazione
- Responsabile della prevenzione e della corruzione
- Il bilancio nella Pubblica Amministrazione
- I termini del procedimento amministrativo
- Il procedimento amministrativo digitale
- L'ufficio relazioni con il pubblico
- La comunicazione della pubblica amministrazione
- La comunicazione sui siti web della pubblica amministrazione
- Il linguaggio della pubblica amministrazione
- La tutela della privacy
- Modalità di presentazione delle domande alla pubblica amministrazione da parte dei cittadini
- Modalità di presentazione delle domande alla pubblica amministrazione da parte delle imprese
- Funzioni della Giunta regionale
- Funzioni del Consiglio regionale
- Differenze tra leggi e regolamenti
- Quando sussiste il dovere di astensione per il pubblico dipendente
- Disciplina generale del procedimento amministrativo
- Informazioni che non possono essere oggetto di pubblicazione
- Obiettivi della performance
- Cos'è il Piano triennale per l'informatica di AGID e perché è importante?
- Nella progettazione di servizi digitali per la P.A. vale il principio digital & mobile first, cosa significa?
- Nella progettazione di servizi digitali per la P.A. vale il principio digital identity only, cosa significa?
- Nella progettazione di servizi digitali per la P.A. vale il principio cloud first, cosa significa?
- La progettazione di servizi digitali per la P.A. deve fornire servizi inclusivi e accessibili, cosa significa?
- Perché nel Piano triennale si dice che i dati pubblici sono un bene comune?
- Nella progettazione di servizi digitali per la P.A. vale il principio interoperabile by design, cosa significa?
- Nella progettazione di servizi digitali per la P.A. vale il principio sicurezza e privacy by design, cosa significa?

- Lo sviluppo di servizi digitali per la P.A. deve avvenire in modo user-centric, data driven e agile, cosa significa?
- Lo sviluppo di servizi digitali per la P.A. deve avvenire nel rispetto del principio di derivazione comunitaria cosiddetto ONCE ONLY, cosa significa?
- Cosa significa Transfrontaliero by design?
- Le pubbliche amministrazioni devono utilizzare il codice aperto quando sviluppano un sw?
- Cos'è e a cosa servono gli strumenti presenti nel sito accessibilita.agid.gov.it?
- Cosa si intende per accessibilità di un sistema informatico?
- Quali sono le funzioni principali il Responsabile per la Transizione Digitale
- Indicatori sul livello di digitalizzazione degli Stati Europei ad esempio il DESI?
- Quali sono gli strumenti per l'analisi statistica dei siti web nella PA?
- Cosa rappresenta l'UX design per un sito web?
- Cosa rappresenta l'UI Kit Italia per un sito web?
- C'è differenza tra UX e UI?
- Nel progettare un servizio on line mediante metodologia Waterfall caratteristiche vantaggi e svantaggi?
- Nel progettare un servizio on line mediante metodologia Agile (Scrum) caratteristiche vantaggi e svantaggi?
- Quando usare metodologia Waterfall o Agile?
- Cosa significa approccio API-first?
- A cosa servono le linee guida per l'interoperabilità tecnica delle PPAA?
- A cosa serve la Piattaforma notifiche digitali?
- Cos'è il Polo strategico nazionale?
- Cos'è la piattaforma Pagopa?
- Cos'è il fascicolo sanitario elettronico?
- Cos'è e a cosa serve l'appIO?
- Nella progettazione di servizi digitali per la P.A. vale il principio digital identity only, cosa significa?
- Nella progettazione di servizi digitali per la P.A. vale il principio cloud first, cosa significa?
- La progettazione di servizi digitali per la P.A. deve fornire servizi inclusivi e accessibili, cosa significa?
- Nella progettazione di servizi digitali per la P.A. vale il principio interoperabile by design, cosa significa?
- L'anagrafe nazionale della popolazione residente ANPR
- Sistemi di pagamento elettronico nella PA
- Il piano triennale dell'informatico nella Pubblica Amministrazione
- Open data nella PA

- Il cloud nella PA
- Il processo di transizione digitale nella PA
- Sistemi di identificazione digitale in Italia ed Europa
- L'interoperabilità nella PA tra enti pubblici
- Quali principi utilizzare nella progettazione software dei servizi della PA
- Open source e close source nella PA
- Cosa si intende per portali accessibili nella PA
- Direttive sull'utilizzo dei datacenter nella PA
- Sicurezza informatica nella PA
- I documenti informatici nella PA
- Utilizzo della firma digitale e della marca temporale nella PA
- La conservazione documentale nella PA
- Le piattaforme della PA
- Deve progettare un sistema di Gestione personale da dove partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un sistema di Gestione patrimonio immobiliare da dove partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un sistema di Gestione contributi casa da dove partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un sistema di Gestione concorsi da dove partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un sistema di Gestione formazione da dove partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un sistema di Gestione autorizzazioni ambientali da dove partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un sistema di Gestione permessi raccolta funghi da dove partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un sistema di Gestione patentini pesca sportiva fluviale da dove partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare la base dati per la Gestione beni demanio regionale (spiagge). Come lo imposterebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare la base dati per la Gestione attrezzature informatiche. Come lo imposterebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare la base dati per la Gestione personale. Come lo imposterebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare la base dati per la Gestione contributi casa. Come lo imposterebbe e su quali elementi concentrerebbe la sua attenzione?

- Deve progettare la base dati per la Gestione concorsi. Come lo imposterebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare la base dati per la Gestione formazione. Come lo imposterebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare la base dati per la Gestione autorizzazioni ambientali. Come lo imposterebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un'app Tessere carburante. Come partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un'app Prenotazione sale. Come partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un'app Offerte lavoro nelle vicinanze. Come partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un'app Stazione di rifornimento carburanti. Come partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un'app Stazione di ricarica elettrica. Come partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un'app Farmacie ed orari nelle vicinanze. Come partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Deve progettare un'app Segnalazione di problemi sul territorio. Come partirebbe e su quali elementi concentrerebbe la sua attenzione?
- Vuole proporre all'amministrazione una nuova tecnologia Progressive web app o l'uso di framework javascript da introdurre nei sistemi informativi dell'Ente. Come presenterebbe la soluzione al Direttore Generale?
- Vuole proporre all'amministrazione una nuova tecnologia container da introdurre nei sistemi informativi dell'Ente. Come presenterebbe la soluzione al Direttore Generale?
- Vuole proporre all'amministrazione una nuova tecnologia Chatbot da introdurre nei sistemi informativi dell'Ente. Come presenterebbe la soluzione al Direttore Generale?
- Vuole proporre all'amministrazione una nuova tecnologia low-code no-code da introdurre nei sistemi informativi dell'Ente. Come presenterebbe la soluzione al Direttore Generale?
- Vuole proporre all'amministrazione una nuova tecnologia microservizi da introdurre nei sistemi informativi dell'Ente. Come presenterebbe la soluzione al Direttore Generale?
- Vuole proporre all'amministrazione una nuova tecnologia Intelligenza artificiale per l'analisi dei dati da introdurre nei sistemi informativi dell'Ente. Come presenterebbe la soluzione al Direttore Generale?
- Vuole proporre all'amministrazione una nuova tecnologia Identità digitale da introdurre nei sistemi informativi dell'Ente. Come presenterebbe la soluzione al Direttore Generale?
- Vuole proporre all'amministrazione una nuova tecnologia blockchain da introdurre nei sistemi informativi dell'Ente. Come presenterebbe la soluzione al Direttore Generale?
- Simply stated, a compiler is a program that reads a program written in one language - the source language - and translates it into an equivalent program in another language - the target language.

As an important part of this transalation process, the compiler reports to its user the presence of errors in the source program.

- There are thousands of source languages, ranging from traditional programming languages such as Fortran and Pascal to specialized languages that have arisen in virtually every area of computer application. Target languages are equally as varied; a target language may be another programming language, or the machine language of any computer between a microprocessor and a supercomputer.
- There are two parts to compilation: analysis and syntesis. The analysis part breaks up the source program into constituent pieces and creates an intermediate representation of the source program. The syntesis part constructs the desired target program from the intermediate representation.
- Compilers are sometimes classified as single-pass, multi-pass, load-and-go, debugging, or optimizing, depending on how they have been constructed or on what function they are supposed to perform. Despite this apparent complexity, the basic tasks that an compiler must perform are essentially the same.
- During analysis, the operation implied by the sourse program are determined and recorded in a hierarchical structure called tree. Often, a special kind of tree called a syntax tree is used, in which each node represents an operation and the children of a node represent the arguments of the operation.
- The structure editor can perform additional tasks that are useful in the preparation of programs. For example, it can check that the input is correctly formed, can supply key-words automatically, and can jump from a begin or left parenthesis to its matching end or right parenthesis.
- Structure editor. A structure editor takes as input a sequence of commands to build a source program. The structure editor not only performs the text creation and modification fuctions of an ordinary text editor, but also analyses the program text, putting an appropriate hierarchical structure on the sourse program.
- Pretty printers. A pretty printer analyses a program and prints it in such a way that the structure of the program becomes clearly visibile. For example, comments may appear in a special font, and statements may appear with an amount of identation proportional to the depth of their nesting in the hierarchical organization of the statements.
- Static checkers. A static checker reads a program, analyzes it, and attempts to discover potential bugs without running the programm. The analysis portion is often similar to that found in optimizing compliers. For example, a static checker may detect that parts of the source program can never be executed, or that a certain variable might be used before being difined. In addition, it can catch logical errors.
- Text formatters. A text formatter takes input that is a stream of characters, most of which is text to be typeset, but some of which includes command indicate paragraphs, figures, or mathematiacal structures like subscripts and superscripts.
- Silicon compilers. A silicon compiler has a source language that is similar or identical to a conventional programming language. However, the variables of the language represents, not location in memory, but logical signals (0 or 1) or groups of signals in a switching circuit. The output is a circuit design in an appropriate language.
- In addition to a compiler, serveral other programs may be required to create an executable target program. A source program may be divided into modules stored in separate files. The task of collection the source program is sometimes entrusted to a dinstinct program, called preprocessor.

- In compiling analysis consist of three phases:
  1) Linear analysis, in which the stream of characters making up the source program is read from left-to-right and grouped into tokens that are sequences of characters having a collective meaning.
- 2) Hierarchical analysis, in which characters or tokens are grouped hierarchically into nested collections with collective meaning.
  3) semantic analysis, in which certains checks are performed to ensure that the components of a program fit togheter meaningfully.
- Syntax Analysis. Hierarchical analysis is called parsing or syntax analysis. It involves grouping the tokens of the source program into grammatical phrases that are used by the compiler to synthesize output. Usually, the grammatical phrases or the source program are represented by a parse tree.
- The division between lexical and syntactical analysis is somewhat arbitrary. We usually choose a division that simplifies the overall task of analysis. One factor in determining the division is whether a source language contruct is inherently recursive or not.
- We would normally reconize identifiers by a simple scan of the input stream, waiting until a character tat was neither a letter nor a digit was found, and then grouping all the letters and digits found up to that point into a identifier token.
- The parse tree describes the syntactic structure of the input. A more common internal representation of the syntactic structure is given by the parse tree in which the operators appear as the interior node, and the operands of an operator are the children of the node for that operator
- A symbol table is a data structure containing a record for each identifier, with fields for the attributes of the identifier. The data structure allows us to find the record for each identifier quickly and to store or retrive data from that record quickly.
- Error detection and reporting. Each phase can encounter errors. However, after detecting an error, a phase must somehow deal with that error, so that compilation can proceed, allowing further errors in the source program to be detected. A compiler that stops when if finds the first error in not helpful as it could be.
- Attributes fot Tokens. When more than one pattern matches a lexeme, the lexical analyzer must provide additional information about the particular lexeme that matches to the subsequent phases of the compiler. For example, the pattern num macheches both the trings 0 and 1, but it is essential for the code generator to know what string was actually matched.
- Buffer pairs. For many source languages, there are times when the lexical analyzer needs to look ahead several characters beyond the lexeme for a pattern before a match can be annouced. Because a large amount of time can be consumed moving characters, specialized buffering techniques have been developed.
- There are three general approches to the implementation of a lexical analyzer: 1 use a lexical-analyzer generator, such as the Lex compiler discussed in the section 3.5, to produce the lexical analyzer from a regualr-expression-based specification. In this case the generator provides routines for reading and buffering the input.
- Implementing a Transition Diagram. A sequence of trnssition diagrams can be converted into a program to look for the tokens specified by the diagrams. We adopt a systematic approach that works for all transition diagrams and constructs programs whose size is pro portional to the number of states and edges in the diagrams.

- A compiler must check that the source program follows both the syntactic and semantic conventions of the source language. This checking, called static checking ensures that certain kinds of programming errors wil be detected and reported.
- A type system is a collection of rules for assigning type expressions to the various parts of a program. A type checker implements a type system. Different type systems may be used by different compilersor processors of the same language.
- Static and Dynamic Checking of Types. Checking done by a compiler is said to be static, while checking done when the trget program runs is termed dynamic. In principe, any check can be done dynamically, if the target code carries the type of an element along with the value of that element.
- The flow of control in a program correspond to a depth-first traversal of the activation tree that starts at the root, visit a node before its children, and recursively visit children at each node in a left-right order.
- In a static allocation, names are bound to storage as the program is compiled, so there is no need for run-time support package. Fortran was designed to permit static storge allocation. A Fortran program consists of a main program, subroutines, and functions (call them all procedures).
- The difference between triples and quadruples may be regarded as a matter of how much indirection is present in the representation. When we ultimately produce target code, each name, temporary of programmed-defined, will be assigned some run-time memory location.

7.9 STORAGE ALLOCATION IN FORTRAN

Fortran was designed to permit static storage allocation, as in Section 7.3. However, there are some issues, such as the treatment of COMMON and EQUIVALENCE declarations, that are fairly special to Fortran. A Fortran compiler can create a number of *data areas*, i.e., blocks of storage in which the values of objects can be stored.

The compiler must compute the size of each data area. For the data areas of the procedures, a single counter suffices, since their sizes are known after each procedure is processed. For COMMON blocks, a record for each block must be kept during the processing of all procedures, since each procedure using a block may have its own idea of how big the block is, and the actual size is the maximum of the sizes implied by the various procedures. If pro-

For each data area the compiler creates a *memory map*, which is a description of the contents of the area. This "memory map" might simply consist of an indication, in the symbol-table entry for each name in the area, of its offset in the area. We need not necessarily have an easy way of answering the question, "What are all the names in this data area?"

In practice, in an assembly language, one and only one name in the set would have an actual location defined by a pseudo-operation, and this name would be made the leader. We trust the reader can see how to modify the algorithm to make one particular name be the leader.

A Simple Equivalence Algorithm

The first algorithms for processing equivalence statements appeared in assemblers rather than compilers. Since these algorithms can be a bit complex, especially when interactions between COMMON and EQUIVALENCE statements are considered, let us treat first a situation typical of an assembly language, where the only EQUIVALENCE statements are of the form

```
EQUIVALENCE A,B+offset
```

In Fortran, however, it is the compiler's job to determine storage locations, so an equivalence set not in COMMON may be viewed as "floating" until the compiler determines the position of the whole set in its appropriate data area. To do so correctly, the compiler needs to know the extent of the equivalence set, that is, the number of locations which the names in the set collectively occupy.

Since there are three fields (*low*, *high*, and a pointer to a COMMON block) that must be associated with each leader, we do not want to allocate space for these fields in all symbol-table entries. One course of action is to use the *parent* field from Algorithm 7.1 to point, in the case of the leader, to a record in a new table with three fields, *low*, *high*, and *comblk*.

We assume that prior to code generation the front end has scanned, parsed, and translated the source program into a reasonably detailed intermediate representation, so the values of names appearing in the intermediate language can be represented by quantities that the target machine can directly manipulate (bits, integers, reals, pointers, etc.).

In the analysis-synthesis model of a compiler, the front end translates a source program into an intermediate representation from which the back end generates target code. Details of the target language are confined to the back end, as far as possible.

Although a source program can be translated directly into the target language, some benefits of using a machine-independent intermediate form are:

1. Retargeting is facilitated; a compiler for a different machine can be created by attaching a back end for the new machine to an existing front end.

A more important benefit of quadruples appears in an optimizing compiler, where statements are often moved around. Using the quadruple notation, the symbol table interposes an extra degree of indirection between the computation of a value and its use.

8.2 DECLARATIONS

As the sequence of declarations in a procedure or block is examined, we can lay out storage for names local to the procedure. For each local name, we create a symbol-table entry with information like the type and the relative address of the storage for the name.

Indirect Triples

Another implementation of three-address code that has been considered is that of listing pointers to triples, rather than listing the triples themselves. This implementation is naturally called indirect triples.

Declarations in a Procedure

The syntax of languages such as C, Pascal, and Fortran, allows all the declarations in a single procedure to be processed as a group. In this case, a global variable, say *offset*, can keep track of the next available relative address.

In the translation scheme of Fig. 8.11 nonterminal P generates a sequence of declarations of the form **id** : T.

When the front end generates addresses, it may have a target machine in mind. Suppose that addresses of consecutive integers differ by 4 on a byte-addressable machine. The address calculations generated by the front end may therefore include multiplications by 4.

The requirements traditionally imposed on a code generator are severe. The output code must be correct and of high quality, meaning that it should make effective use of the resources of the target machine. Moreover, the code generator itself should run efficiently.

Input to the Code Generator

The input to the code generator consists of the intermediate representation of the source program produced by the front end, together with information in the symbol table that is used to determine the run-time addresses of the data objects denoted by the names in the intermediate representation.

As we noted in the previous chapter, there are several choices for the intermediate language, including: linear representations such as postfix notation, three-address representations such as quadruples, virtual machine representations such as stack machine code, and graphical representations such as syntax trees and dags.