INTERNATIONAL STANDARD

NORME INTERNATIONALE

ISO/IEC 2382-15

First edition
Première édition
1999-06-15

Information technology — Vocabulary —

Part 15:
Programming languages

Technologies de l'information — Vocabulaire —

Partie 15:
Langages de programmation

Reference number
Numéro de référence
ISO/IEC 2382-15:1999(E/F)

Contents

Sommaire

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.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. 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.

International Standard ISO/IEC 2382-15 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 1, *Vocabulary*.

This first edition cancels and replaces ISO 2382-15:1985, which has been technically revised.

ISO/IEC 2382 will consist of some 37 parts, under the general title *Information technology – Vocabulary*.

Avant-propos

L'ISO (Organisation internationale de normalisation) et la CEI (Commission électrotechnique internationale) forment ensemble un système consacré à la normalisation internationale considérée comme un tout. Les organismes nationaux membres de l'ISO ou de la CEI participent au développement de Normes internationales par l'intermédiaire des comités techniques créés par l'organisation concernée afin de s'occuper des différents domaines particuliers de l'activité technique. Les comités techniques de l'ISO et de la CEI collaborent dans des domaines d'intérêt commun. D'autres organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'ISO et la CEI participent également aux travaux.

Les Normes internationales sont rédigées conformément aux règles données dans les Directives ISO/CEI, Partie 3.

Dans le domaine des technologies de l'information, l'ISO et la CEI ont créé un comité technique mixte, l'ISO/CEI JTC 1. Les projets de Normes internationales adoptés par le comité technique mixte sont soumis aux organismes nationaux pour approbation, avant leur acceptation comme Normes internationales. Les Normes internationales sont approuvées conformément aux procédures qui requiert l'approbation de 75% au moins des organismes nationaux votants.

La Norme internationale ISO/CEI 2382-15 a été élaborée par le comité technique mixte ISO/CEI JTC 1, *Technologies de l'information*, sous-comité SC 1, *Vocabulaire*.

Cette première édition annule et remplace l'ISO 2382-15:1985, dont elle constitue une révision technique.

L'ISO/CEI 2382 comprendra environ 37 parties, présentées sous le titre général *Technologies de l'information – Vocabulaire*.

Introduction

Information technology gives rise to numerous international exchanges of both an intellectual and a material nature. These exchanges often become difficult, either because of the great variety of terms used in various fields or languages to express the same concept, or because of the absence or imprecision of the definitions of useful concepts.

To avoid misunderstandings and to facilitate such exchanges it is essential to clarify the concepts, to select terms to be used in various languages or in various countries to express the same concept, and to establish definitions providing satisfactory equivalents for the various terms in different languages.

ISO 2382 was initially based mainly on the usage to be found in the *Vocabulary of Information Processing* which was established and published by the International Federation for Information Processing and the International Computation Centre, and in the *American National Dictionary for Information Processing Systems* and its earlier editions published by the American National Standards Institute (formerly known as the American Standards Association). Published and Draft International Standards relating to information technology of other international organizations (such as the International Telecommunication Union and the International Electrotechnical Commission) as well as published and draft national standards have also been considered.

The purpose of ISO/IEC 2382 is to provide definitions that are rigorous, uncomplicated and which can be understood by all concerned. The scope of each concept defined has been chosen to provide a definition that is suitable for general application. In those circumstances, where a restricted application is concerned, the definition may need to be more specific.

However, while it is possible to maintain the self-consistency of individual parts, the reader is warned that the dynamics of language and the problems associated with the standardization and maintenance of vocabularies may introduce duplications and inconsistencies among parts.

Introduction

Les technologies de l'information sont à l'origine de multiples échanges intellectuels et matériels sur le plan international. Ceux-ci souffrent souvent de difficultés provoquées par la diversité des termes utilisés pour exprimer la même notion dans des langues ou des domaines différents ou encore de l'absence ou de l'imprécision des définitions pour les notions les plus utiles.

Pour éviter des malentendus et faciliter de tels échanges, il paraît essentiel de préciser les notions, de choisir les termes à employer dans les différentes langues et dans les divers pays pour exprimer la même notion, et d'établir pour ces termes des définitions équivalentes dans chaque langue.

L'ISO 2382 a été basée à l'origine principalement sur l'usage tel qu'il a été relevé, d'une part, dans le *Vocabulary of Information Processing* établi et publié par l'International Federation for Information Processing et le Centre international de calcul et, d'autre part, dans l'*American National Dictionary for Information Processing Systems* y compris ses éditions précédentes publiées par l'American National Standards Institute (connu auparavant sous l'appellation d'American Standards Association). Les Normes internationales publiées ou au stade de projets concernant les technologies de l'information émanant d'autres organisations internationales (telles que l'Union internationale des télécommunications et la Commission électrotechnique internationale) ainsi que les normes nationales publiées ou au stade de projets, ont également été prises en compte.

Le but de l'ISO/CEI 2382 est de procurer des définitions rigoureuses, simples et compréhensibles pour tous les intéressés. La portée de chaque notion a été choisie de façon que sa définition puisse avoir la valeur la plus générale. Cependant, il est parfois nécessaire de restreindre une notion à un domaine plus étroit et de lui donner alors une définition plus spécifique.

D'autre part, si l'on peut assurer la cohérence interne de chaque partie prise individuellement, la cohérence des diverses parties entre elles est plus difficile à atteindre. Le lecteur ne doit pas s'en étonner: la dynamique des langues et les problèmes de l'établissement et de la révision des normes de vocabulaire peuvent être à l'origine de quelques répétitions ou contradictions entre des parties qui ne sont pas toutes préparées et publiées simultanément.

Information technology – Vocabulary –

Part 15: Programming languages

Section 1: General

1.1 Scope

This part of ISO/IEC 2382 is intended to facilitate international communication in information technology. It presents, in two languages, terms and definitions of selected concepts relevant to the field of information technology and identifies relationships among the entries.

In order to facilitate their translation into other languages, the definitions are drafted so as to avoid, as far as possible, any peculiarity attached to a language.

This part of ISO/IEC 2382 defines concepts related to programming languages.

1.2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 2382. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 2382 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 2382-1:1993, *Information technology – Vocabulary – Part 1: Fundamental terms.*

ISO 2382-2:1976, *Data processing – Vocabulary – Part 02: Arithmetic and logic operations.*

ISO/IEC 2382-7:—[1)], *Information technology – Vocabulary – Part 7: Computer programming.*

1) To be published. (Revision of ISO/IEC 2382-7:1989)

Technologies de l'information – Vocabulaire –

Partie 15: Langages de programmation

Section 1: Généralités

1.1 Domaine d'application

La présente partie de l'ISO/CEI 2382 a pour objet de faciliter les échanges internationaux dans le domaine des technologies de l'information. À cet effet, elle présente un ensemble bilingue de termes et de définitions ayant trait à des notions choisies dans ce domaine, et définit les relations pouvant exister entre les différentes notions.

Les définitions ont été établies de manière à éviter les particularismes propres à une langue donnée, en vue de faciliter leur transposition dans les langues autres que celles ayant servi à la rédaction initiale.

La présente partie de l'ISO/CEI 2382 définit les différentes notions relatives aux langages de programmation.

1.2 Références normatives

Les documents normatifs suivants contiennent des dispositions qui, par suite de la référence qui y est faite, constituent des dispositions valables pour la présente partie de l'ISO/CEI 2382. Pour les références datées, les amendements ultérieurs ou les révisions de ces publications ne s'appliquent pas. Toutefois, les parties prenantes des accords fondés sur la présente partie de l'ISO/CEI 2382 sont invitées à rechercher la possibilité d'appliquer les éditions les plus récentes des documents normatifs indiqués ci-après. Pour les références non datées, la dernière édition du document normatif en référence s'applique. Les membres de l'ISO et de la CEI possèdent le registre des Normes internationales en vigueur.

ISO/CEI 2382-1:1993, *Technologies de l'information – Vocabulaire – Partie 1: Termes fondamentaux.*

ISO 2382-2:1976, *Traitement de l'information – Vocabulaire – Partie 02: Opérations arithmétiques et logiques.*

ISO/CEI 2382-7:—[1)], *Technologies de l'information – Vocabulaire – Partie 7: Programmation des ordinateurs.*

1) À publier. (Révision de l'ISO/CEI 2382-7:1989)

1.3 Principles and rules followed

1.3.1 Definition of an entry

Section 2 comprises a number of entries. Each entry consists of a set of essential elements that includes an index number, one term or several synonymous terms, and a phrase defining one concept. In addition, an entry may include examples, notes or illustrations to facilitate understanding of the concept.

Occasionally, the same term may be defined in different entries, or two or more concepts may be covered by one entry, as described in 1.3.5 and 1.3.8 respectively.

Other terms such as **vocabulary**, **concept**, **term**, and **definition** are used in this part of ISO/IEC 2382 with the meaning defined in ISO 1087.

1.3.2 Organization of an entry

Each entry contains the essential elements defined in 1.3.1 and, if necessary, additional elements. The entry may contain the following elements in the following order:

a) an index number (common for all languages in which this part of ISO/IEC 2382 is published) ;

b) the term or the generally preferred term in the language. The absence of a generally preferred term for the concept in the language is indicated by a symbol consisting of five dots (.....); a row of dots may be used to indicate, in a term, a word to be chosen in each particular case ;

c) the preferred term in a particular country (identified according to the rules of ISO 3166) ;

d) the abbreviation for the term ;

e) permitted synonymous term(s);

f) the text of the definition (see 1.3.4);

g) one or more examples with the heading "Example(s)";

h) one or more notes specifying particular cases in the field of application of the concepts with the heading "NOTE(S)";

i) a picture, a diagram, or a table which could be common to several entries.

1.3 Principes d'établissement et règles suivies

1.3.1 Définition de l'article

La section 2 est composée d'un certain nombre d'articles. Chaque article est composé d'un ensemble d'éléments essentiels comprenant le numéro de référence, le terme ou plusieurs termes synonymes et la définition de la notion couverte par ces termes. Cet ensemble peut être complété par des exemples, des notes, des schémas ou des tableaux destinés à faciliter la compréhension de la notion.

Parfois, le même terme peut être défini dans des articles différents, ou bien deux notions ou davantage peuvent être couvertes par un seul article: voir respectivement en 1.3.5 et 1.3.8.

D'autres termes tels que **vocabulaire**, **notion**, **terme**, **définition**, sont employés dans la présente partie de l'ISO/CEI 2382 avec le sens qui leur est donné dans l'ISO 1087.

1.3.2 Constitution d'un article

Chaque article contient des éléments essentiels définis en 1.3.1 et, si nécessaire, des éléments supplémentaires. L'article peut donc comprendre dans l'ordre les éléments suivants:

a) un numéro de référence (le même, quelle que soit la langue de publication de la présente partie de l'ISO/CEI 2382) ;

b) le terme, ou le terme préféré en général dans la langue. L'absence, dans une langue, de terme consacré ou à conseiller pour exprimer une notion est indiquée par un symbole consistant en cinq points de suspension (.....) ; les points de suspension peuvent être employés pour désigner, dans un terme, un mot à choisir dans un cas particulier ;

c) le terme préféré dans un certain pays (identifié selon les règles de l'ISO 3166) ;

d) l'abréviation pouvant être employée à la place du terme ;

e) le terme ou les termes admis comme synonymes ;

f) le texte de la définition (voir 1.3.4) ;

g) un ou plusieurs exemples précédés du titre « Exemple(s) » ;

h) une ou plusieurs notes précisant le domaine d'application de la notion, précédées du titre « NOTE(S) » ;

i) une figure, un schéma ou un tableau, pouvant être communs à plusieurs articles.

1.3.3 Classification of entries

A two-digit serial number is assigned to each part of ISO/IEC 2382, beginning with 01 for "Fundamental terms".

The entries are classified in groups to each of which is assigned a four-digit serial number; the first two digits being those of the part of ISO/IEC 2382.

Each entry is assigned a six-digit index number; the first four digits being those of the part of ISO/IEC 2382 and the group.

To show the relationship between versions of ISO/IEC 2382 in various languages, the numbers assigned to parts, groups, and entries are the same for all languages.

1.3.4 Selection of terms and wording of definitions

The selection of terms and the wording of definitions have, as far as possible, followed established usage. Where there were contradictions, solutions agreeable to the majority have been sought.

1.3.5 Multiple meanings

When, in one of the working languages, a given term has several meanings, each meaning is given a separate entry to facilitate translation into other languages.

1.3.6 Abbreviations

As indicated in 1.3.2, abbreviations in current use are given for some terms. Such abbreviations are not used in the texts of the definitions, examples or notes.

1.3.7 Use of parentheses

In some terms, one or more words printed in bold typeface are placed between parentheses. These words are part of the complete term, but they may be omitted when use of the abridged term in a technical context does not introduce ambiguity. In the text of another definition, example, or note of ISO/IEC 2382, such a term is used only in its complete form.

In some entries, the terms are followed by words in parentheses in normal typeface. These words are not a part of the term but indicate directives for the use of the term, its particular field of application, or its grammatical form.

1.3.3 Classification des articles

Chaque partie de l'ISO/CEI 2382 reçoit un numéro d'ordre à deux chiffres, en commençant par 01 pour la partie « Termes fondamentaux ».

Les articles sont répartis en groupes qui reçoivent chacun un numéro d'ordre à quatre chiffres, les deux premiers chiffres étant ceux du numéro de la partie de l'ISO/CEI 2382.

Chaque article est repéré par un numéro de référence à six chiffres, les quatre premiers chiffres étant ceux du numéro de partie de l'ISO/CEI 2382 et de groupe.

Les numéros des parties, des groupes et des articles sont les mêmes pour toutes les langues, afin de mettre en évidence les correspondances des versions de l'ISO/CEI 2382.

1.3.4 Choix des termes et des définitions

Les choix qui ont été faits pour les termes et leurs définitions sont, dans toute la mesure du possible, compatibles avec les usages établis. Lorsque certains usages apparaissent contradictoires, des solutions de compromis ont été retenues.

1.3.5 Pluralité de sens ou polysémie

Lorsque, dans l'une des langues de travail, un même terme peut prendre plusieurs sens, ces sens sont définis dans des articles différents, pour faciliter l'adaptation du vocabulaire dans d'autres langues.

1.3.6 Abréviations

Comme indiqué en 1.3.2, des abréviations d'usage courant, au moins en anglais, sont indiquées pour certains termes. De telles abréviations ne sont pas employées dans le corps des définitions, exemples ou notes.

1.3.7 Emploi des parenthèses

Dans certains termes, un ou plusieurs mots imprimés en caractères gras sont placés entre parenthèses. Ces mots font partie intégrante du terme complet, mais peuvent être omis lorsque le terme ainsi abrégé peut être employé dans un contexte technique déterminé sans que cette omission ne crée d'ambiguïté. Un tel terme n'est employé dans le texte d'une autre définition, d'un exemple ou d'une note de l'ISO/CEI 2382, que sous sa forme complète.

Dans certains articles, les termes définis sont suivis par des expressions imprimées en caractères normaux et placées entre parenthèses. Ces expressions ne font pas partie du terme mais indiquent des prescriptions d'emploi, précisent un domaine d'application particulier ou indiquent une forme grammaticale.

1.3.8 Use of brackets

When several closely related terms can be defined by texts that differ only in a few words, the terms and their definitions are grouped in a single entry. The words to be substituted in order to obtain the different meanings are placed in brackets, i.e. [ ], in the same order in the term and in the definition. To clearly identify the words to be substituted, the last word that according to the above rule could be placed in front of the opening bracket is, wherever possible, placed inside the bracket and repeated for each alternative.

1.3.9 Use of terms printed in italic typeface in definitions and the use of an asterisk

A term printed in italic typeface in a definition, an example, or a note is defined in another entry in ISO/IEC 2382, which may be in another part. However, the term is printed in italic typeface only the first time it occurs in each entry.

Italic typeface is also used for other grammatical forms of a term, for example, plurals of nouns and participles of verbs.

The basic forms of all terms printed in italic typeface which are defined in this part of ISO/IEC 2382 are listed in the index at the end of the part (see 1.3.11).

An asterisk is used to separate terms printed in italic typeface when two such terms are referred to in separate entries and directly follow each other (or are separated only by a punctuation mark).

Words or terms that are printed in normal typeface are to be understood as defined in current dictionaries or authoritative technical vocabularies.

1.3.10 Spelling

In the English language version of this part of ISO/IEC 2382, terms, definitions, examples, and notes are given in the spelling preferred in the USA. Other correct spellings may be used without violating this part of ISO/IEC 2382.

1.3.11 Organization of the alphabetical index

For each language used, an alphabetical index is provided at the end of each part. The index includes all terms defined in the part. Multiple-word terms appear in alphabetical order under each of their key words.

1.3.8 Emploi des crochets

Lorsque plusieurs termes étroitement apparentés peuvent être définis par des textes presque identiques, à quelques mots près, les termes et leurs définitions ont été groupés en un seul article. Les mots à substituer à ceux qui les précèdent pour obtenir les différents sens sont placés entre crochets (c'est-à-dire [ ]) dans le même ordre dans le terme et la définition. En vue d'éviter toute incertitude sur les mots à remplacer, le dernier mot qui, suivant la règle ci-dessus, pourrait être placé devant le crochet d'ouverture, est placé, si possible, à l'intérieur des crochets et répété à chaque occasion.

1.3.9 Emploi dans les définitions de termes imprimés en caractères italiques et de l'astérisque

Dans le texte d'une définition, d'un exemple ou d'une note, tout terme imprimé en caractères italiques a le sens défini dans un autre article de l'ISO/CEI 2382, qui peut se trouver dans une autre partie. Cependant le terme est imprimé en caractères italiques uniquement la première fois qu'il apparaît dans chaque article.

Les caractères italiques sont également utilisés pour les autres formes grammaticales du terme, par exemple, les noms au pluriel et les verbes au participe.

La liste des formes de base des termes imprimés en caractères italiques qui sont définis dans cette partie de l'ISO/CEI 2382 est fournie dans l'index à la fin de la partie (voir 1.3.11).

L'astérisque sert à séparer les termes imprimés en caractères italiques quand deux termes se rapportent à des articles séparés et se suivent directement (ou bien sont séparés simplement par un signe de ponctuation).

Les mots ou termes imprimés en caractères normaux doivent être compris dans le sens qui leur est donné dans les dictionnaires courants ou vocabulaires techniques faisant autorité.

1.3.10 Mode d'écriture et orthographe

Dans la version anglaise de la présente partie de l'ISO/CEI 2382, les termes, définitions, exemples et notes sont écrits suivant l'orthographe prévalant aux États-Unis. D'autres orthographes correctes peuvent être utilisées sans violer la présente partie de l'ISO/CEI 2382.

1.3.11 Constitution de l'index alphabétique

Pour chaque langue de travail, un index alphabétique est fourni à la fin de chaque partie. L'index comprend tous les termes définis dans la partie. Les termes composés de plusieurs mots sont répertoriés alphabétiquement suivant chacun des mots clés.

Section 2: Terms and definitions

15 Programming languages

15.01 Lexical tokens

15.01.01
lexical token
lexical element
lexical unit
A *string* of one or more *characters* of the *alphabet* of a *programming language* that, by convention, represents an elemental unit of meaning.

Examples: A *literal* such as 2G5 or an *identifier* such as last_name in Pascal.

15.01.02
language construct
A syntactically allowable part of a *program* that may be formed from one or more *lexical tokens* in accordance with the rules of a *programming language*.

15.01.03
identifier (in programming languages)
A *lexical token* that names a *language construct*.

Examples: The names of *variables*, *arrays*, *records*, *labels*, *procedures*, etc.

NOTE – An identifier usually consists of a *letter* optionally followed by letters, *digits,* or other *characters*.

15.01.04
predefined identifier
An *identifier* that is defined as part of a *programming language*.

Example: *A reserved word*.

NOTE – If a predefined identifier is not reserved, then a *declaration* using that identifier redefines its meaning for the *scope* of the declaration.

15.01.05
reserved word
A *predefined identifier* that cannot be redefined by a programmer.

NOTE – Not all *programming languages* have reserved words.

15.01.06
delimiter (in programming languages)
separator (deprecated in this sense)
A *lexical token* that indicates the beginning or the end of another lexical token or of a *character string* considered as a syntactic unit.

NOTES
1. *Special characters* or *reserved words* may serve as delimiters.
2. Contrast with *separator*.

Section 2: Termes et définitions

15 Langages de programmation

15.01 Unités lexicales

15.01.01
unité lexicale
entité lexicale
Chaîne d'un ou plusieurs *caractères* de l'*alphabet* d'un *langage de programmation* représentant, par convention, une unité sémantique élémentaire.

Exemples: Un *libellé* tel que 2G5 ou un *identificateur* tel que last_name en Pascal.

15.01.02
élément de langage
Partie, autorisée par la syntaxe, d'un *programme*, formée d'une ou plusieurs *unités lexicales* et utilisée conformément aux règles d'un *langage de programmation*.

15.01.03
identificateur (en langages de programmation)
Unité lexicale désignant un *élément de langage*.

Exemples: Noms de *variables*, de *tableaux*, d'*articles*, d'*étiquettes*, de *procédures*.

NOTE – Un identificateur est souvent constitué d'une *lettre*, suivie éventuellement de lettres, de *chiffres* ou d'autres *caractères*.

15.01.04
identificateur prédéfini
Identificateur défini comme faisant partie d'un *langage de programmation*.

Exemple: *Mot réservé*.

NOTE – Si un identificateur prédéfini n'est pas réservé, une *déclaration* qui utilise cet identificateur redéfinit son sens pour la *portée* de la déclaration.

15.01.05
mot réservé
Identificateur prédéfini qui ne peut être redéfini par un programmeur.

NOTE – Certains *langages de programmation* n'ont aucun mot réservé.

15.01.06
délimiteur (en langages de programmation)
Unité lexicale indiquant le début ou la fin d'une autre unité lexicale ou d'une chaîne de caractères considérée comme une unité syntaxique.

NOTES
1. Des *caractères spéciaux* ou des *mots réservés* peuvent servir de délimiteurs.
2. Comparer avec *séparateur*.

15.01.07
separator
A *delimiter* that prevents adjacent *lexical tokens* or syntactic units from being interpreted as a single item.

Examples: The *space character* or a *format effector*.

NOTE – Contrast with *delimiter.*

15.01.08
to overload
To assign more than one meaning to a *lexical token.*

Example: The lexical token "+" can mean *integer* addition, real addition, set union, concatenation, etc.

15.01.09
disambiguation
The action of determining which *language construct*, of several with the same *sequence* of *lexical tokens*, is referred to by a particular occurrence within a *program*.

15.01.10
label (in programming languages)
An *identifier* for a location in a *program*.

NOTES
1. A label is frequently used to refer to a *statement.*
2. In BASIC, a line number can serve as a label, but is not always the target of a transfer.
3. In Fortran, a label, consisting of up to five *digits,* that precedes a statement, may be used to refer to the statement.

15.01.11
comment
remark
A *language construct* exclusively used to include *text* that has no intended effect on the *execution* of the *program*.

Examples: An explanation to a human reader; *data* for an *automatic* documentation system.

15.02 Declarations

15.02.01
declaration
An explicit *language construct* that introduces one or more *identifiers* into a *program* and specifies how these identifiers are to be interpreted.

Examples: Declarations of *data types*, *storage organization*, *packages*, or *tasks*.

NOTE – In some *programming languages*, declarations are considered to be *statements*.

15.02.02
declarative part
data division
A portion of a *program* that consists of one or more *declarations*.

NOTE – In COBOL, a declarative part is called “data division”.

15.01.07
séparateur
Délimiteur qui évite de considérer comme un élément unique des *unités lexicale*s ou syntaxiques adjacentes.

Exemples: *Caractère espace* ou *caractère de mise en page*.

NOTE – Comparer avec *délimiteur.*

15.01.08
surcharger
Donner plusieurs significations à une *unité lexicale*.

Exemple: L'unité lexicale « + » peut signifier une addition d'*entiers relatifs*, une addition de *nombres réels*, une union d'ensembles, une concaténation, etc.

15.01.09
désambiguïsation
Détermination, parmi plusieurs *éléments de langage* ayant la même *séquence* d’*unités lexicales*, de celui qui est désigné par une occurrence particulière dans un *programme*.

15.01.10
étiquette (en langages de programmation)
Identificateur pour un endroit dans un *programme*.

NOTES
1. Une étiquette est souvent utilisée pour se référer à une *instruction.*
2. En BASIC, un numéro de ligne peut servir d'étiquette mais ce n'est pas toujours la cible d'un transfert
3. En Fortran, une étiquette d'un maximum de 5 *chiffres* précédant une instruction peut être utilisée pour se référer à cette instruction.

15.01.11
commentaire
Élément de langage utilisé exclusivement pour insérer du *texte*, mais qui n'est pas destiné à avoir d'effet sur l'*exécution* du *programme*.

Exemples: Une explication pour une personne ; des *données* pour un système de documentation *automatique*.

15.02 Déclarations

15.02.01
déclaration
Élément de langage explicite qui introduit un ou plusieurs *identificateurs* dans un *programme* et qui indique comment les interpréter.

Exemples: Déclarations qui désignent *des types de données*, l’*organisation de mémoire*, des *paquetages* ou des *tâches*.

NOTE – Dans certains *langages de programmation*, on considère les déclarations comme des *instructions*.

15.02.02
partie déclarative
Portion d'un *programme* constituée d'une ou plusieurs *déclarations*.

NOTE – En COBOL, une partie déclarative s’appelle « division données ».

15.02.03
default (adj.)
Pertaining to an *attribute*, *data value,* or option that is assumed when none is explicitly specified.

Example: In Fortran, the default naming convention specifies that names beginning with one of the *letters* I through N denote *variables* of *integer type*.

15.02.04
implicit declaration
A *declaration* caused by the occurrence of an *identifier* that designates an *object*, whose characteristics are determined by default.

Example: In Pascal "output = text".

15.02.05
predefined
built-in
intrinsic
Pertaining to a *language construct* that is declared by the definition of the *programming language*.

Examples: The predefined function SIN in PL/I, the predefined *data type* INTEGER in Fortran.

15.02.06
scope
scope of a declaration
That portion of a *program* within which a *declaration* is valid.

15.02.07
shared data
Data that can be *accessed* by two or more *modules* that may be *executed* asynchronously or concurrently.

Examples: COMMON in Fortran; "compool" in some *programming languages*; PL/I single *variables* tagged EXTERNAL; a form of a *package* in Ada.

15.02.08
dynamic scope
The *scope* created by the *activation* of portions or all of the *modules* that contain *declarations* used by another module that lacks these declarations during the *execution* of the latter module.

15.02.09
static scope
The *scope* as determined by finding the innermost surrounding *module* in which the *declaration* is made.

NOTE – *Desk checking* of a *program* is sufficient for finding a static scope.

15.02.03
par défaut (qualificatif)
implicite (qualificatif)
Qualifie un *attribut*, une *valeur de donnée*, ou une option retenus en l'absence d'autre précision.

Exemple: En Fortran, la convention de désignation par défaut précise que les noms qui commencent par une *lettre* de I à N désignent des *variables* de *type entier*.

15.02.04
déclaration implicite
Déclaration provoquée par l'apparition d'un *identificateur* désignant un *objet* dont les caractéristiques sont fournies *par défaut*.

Exemple: En Pascal « sortie = texte ».

15.02.05
prédéfini
intrinsèque
incorporé
Qualifie un *élément de langage* dont la *déclaration* figure dans la définition du *langage de programmation*.

Exemples: La fonction prédéfinie SIN en PL/I, le *type de données* prédéfini INTEGER en Fortran.

15.02.06
portée
portée d'une déclaration
champ d'application d'une déclaration
Partie du *programme* à l'intérieur duquel une *déclaration* est valide.

15.02.07
donnée partagée
Donnée accessible par au moins deux *modules* qui peuvent être *exécutés* de façon asynchrone ou simultanée.

Exemples: Variable de classe COMMON en Fortran ; « compool » dans certains *langages de programmation* ; *variables* simples de PL/I désignées EXTERNAL ; forme de *bloc* en Ada.

15.02.08
portée dynamique
champ d'application dynamique
Portée créée par l'*activation* de portions de *modules* ou de modules entiers qui contiennent des *déclarations* utilisées par un autre module qui n'a pas ces déclarations, pendant l'*exécution* de ce dernier module.

15.02.09
portée statique
champ d'application statique
Portée d'une *déclaration*, déterminée par le *module* englobant le plus intérieur contenant cette déclaration.

NOTE – Le *contrôle manuel* d'un *programme* suffit pour trouver une portée statique.

15.02.10
declarative region
A portion of a *program* consisting of *declarations*.

15.02.11
local (adj.)
Pertaining to a *language construct* that has a *scope* only within the *declarative region* in which it is declared.

15.02.12
global
Pertaining to a *language construct* that is within the *scope* of all *modules* of the *program*.

15.02.13
external
Pertaining to a *language construct* that is defined outside the *module* in which it is referenced.

NOTE – A *declaration* may be required within the module to provide a name and to indicate that the complete definition is external.

15.02.14
static (adj.)
Pertaining to *objects* that exist and retain their values throughout the *execution* of the entire *program*.

Example: A *subprogram * variable* that has been declared static to retain its values from one execution to the next.

15.02.15
dynamic
Pertaining to a *data attribute*, whose values can only be established during the *execution* of all or part of a *program*.

Example: The length of a variable-length *data object* is dynamic.

15.02.16
lifetime
The portion of the *execution duration* during which a *language construct* exists.

15.02.17
visibility (1)
The ability to make a reference to a particular *language construct* at a specific place in a *module*.

15.02.18
visibility (2)
The portion of a *program* within which a reference can be made to a specific *language construct*.

15.03 Data objects

15.03.01
data structure
A physical or logical relationship among units of *data* and the data themselves.

15.02.10
zone déclarative
région déclarative
Portion d'un *programme* composée de *déclarations*.

15.02.11
local
Qualifie un *élément de langage* qui n'a de *portée* qu'à l'intérieur de la *région déclarative* dans laquelle il est déclaré.

15.02.12
global
Qualifie un *élément de langage* qui fait partie de la *portée* de tous les *modules* du *programme*.

15.02.13
externe
Qualifie un *élément de langage* qui est défini en dehors du *module* dans lequel il est cité.

NOTE – Une *déclaration* peut être nécessaire a l'intérieur d'un module afin de fournir un nom et d'indiquer que la définition entière est externe.

15.02.14
statique
Qualifie des *objets* qui existent et conservent leurs valeurs pendant l'*exécution* de tout le *programme*.

Exemple: *Variable* d'un *sous-programme* qui a été déclarée statique afin de conserver ses valeurs d'une exécution à l'autre.

15.02.15
dynamique
Qualifie un *attribut de données* dont les valeurs ne peuvent être déterminées que durant l'*exécution* de tout ou partie d'un *programme*.

Exemple: La longueur d'un *objet de données* de longueur variable est dynamique.

15.02.16
durée de vie
Portion de la *durée d'exécution* pendant laquelle un *élément de langage* existe.

15.02.17
visibilité
Capacité de faire référence à un *élément de langage* particulier depuis un endroit spécifique dans un *module*.

15.02.18
zone de visibilité
Partie d'un *programme* dans laquelle on peut faire référence à un *élément de langage* particulier.

15.03 Objets de données

15.03.01
structure de données
Ensemble constitué de *données* et d'une relation physique ou logique entre elles.

15.03.02
data object (in programming languages)
An element of a *data structure* such as a *file*, an *array*, or an *operand*, that is needed for the *execution* of *programs*.

NOTE – A data object may be a *constant* or a *variable*.

15.03.03
variable
A quadruple, established by a *declaration* or an *implicit declaration*, that consists of an *identifier*, a set of *data attributes*, one or more *addresses*, and *data values*, where the relationship between the addresses and the data values may vary.

NOTE – In some *programming languages*, the addresses may vary, hence the associated data values may vary. In other programming languages, the addresses remain fixed, but the associated data values may change during *execution*.

15.03.04
data value
An element of a declared set of *data objects* that, in a specific context, is associated with a *language construct* such as a *variable* or a *data type*.

NOTE – In principle, the data value should be distinguished from "function value" in mathematics, from "value of a number" and "position value" in *numeric representation*.

15.03.05
constant
A quadruple, established by a de*claration* or an *implicit declaration*, that consists of an *identifier*, a set of *data attributes*, one or more *addresses*, and only one *data value*.

15.03.06
aggregate
A structured collection of components, where the components may have the same or different *data structure*, and where the data structure of the collection itself may also be a constituent part of a corresponding *composite type*.

15.03.07
aggregate value
The *data value* associated with an *aggregate*.

15.03.08
array
An *aggregate* that is an instance of an *array type* and each element or appropriate group of elements in which may be referenced randomly and independently of the others.

15.03.09
array slice
slice
A portion of an *array* that consists of contiguous cells along any dimension.

NOTE – In Ada, an array slice is also a basic *operation*.

15.03.02
objet de données (en langages de programmation)
Élément d'une *structure de données,* tel qu'un *fichier,* un *tableau,* ou un *opérande,* qui est nécessaire pour l'*exécution* de *programmes*.

NOTE – Un objet de données peut être une *constante* ou une *variable*.

15.03.03
variable
Quadruplet, défini par une *déclaration* ou par une *déclaration implicite*, constitué d'un *identificateur*, d'un ensemble d'*attributs de données,* d'une ou de plusieurs *adresses,* et de *valeurs de données,* et dans lequel la relation entre les adresses et les valeurs peut changer.

NOTE – Dans certains *langages de programmation,* les adresses peuvent changer, et donc les valeurs de données associées peuvent changer. Dans d'autres langages de programmation, les adresses restent fixes mais les valeurs de données associées peuvent changer pendant l'*exécution*.

15.03.04
valeur de donnée
Élément d'un ensemble défini d'*objets de données* qui, dans un contexte donné, est associé à un *élément de langage* tel qu'une *variable* ou un *type de données.*

NOTE – En principe, la valeur de donnée devrait être distinguée de « valeur de fonction » en mathématiques, de « valeur de nombre » et de « valeur de position » en *représentation numérique*.

15.03.05
constante
Quadruplet défini par une *déclaration* ou par une *déclaration implicite,* et constitué d'un *identificateur*, d'un ensemble d'*attributs de données,* d'une ou de plusieurs *adresses,* et d'une seule *valeur de donnée.*

15.03.06
agrégat
Collection structurée de composants, dans laquelle les composants peuvent avoir ou non la même *structure de données,* et où la structure de données de la collection elle-même peut aussi être une partie constitutive d'un *type composite* correspondant.

15.03.07
valeur d'agrégat
Valeur de donnée associée à un *agrégat*.

15.03.08
tableau
Agrégat qui est un exemple d'un *type tableau* et dont chaque élément ou groupe approprié d'éléments peut être référencé directement et indépendamment des autres.

15.03.09
tranche de tableau
tranche
Partie d'un *tableau* composée de cellules adjacentes selon une quelconque des dimensions.

NOTE – En Ada, le terme « tranche de tableau » est aussi employé pour désigner une *opération* de base.

15.03.10
variant part
A part of a *record*, composed of *data objects*, whose corresponding *data structures* or declared *data types* may vary.

NOTE – Both the number of data objects and their composition may vary.

15.03.11
variant record
A *record* that contains a *variant part*.

NOTE – The record may contain *discriminants* to indicate the *data types* in the variant part.

15.03.12
discriminant (noun)
A *parameter*-like *language construct* that indicates the *data structure* to be used within a given *variant record*.

15.03.13
parameter (in programming languages)
A *language construct* for passing *data objects* or *data values* between *modules*.

15.03.14
actual parameter
actual argument
A *parameter*, such as an *expression*, * *identifier*, or other *language construct*, used in a *call* or *generic instantiation* for association of a *data object* with a corresponding *declaration*.

NOTE – The corresponding declaration is called *formal parameter*.

15.03.15
formal parameter
dummy argument
A *parameter*, defined in the *declaration* of certain *modules*, that is associated with an *actual parameter* in a *call* or *generic instantiation*.

15.03.16
parameter association
The association between *formal parameters* and their corresponding *actual parameters* in a *call* or *generic instantiation*.

15.03.17
data attribute
A predefined characteristic of a *data type, * data object*, * *module*, or some other *language construct.*

Examples: A *real type* may have the data attribute PRECISION with the *data values* SINGLE or DOUBLE. A *task* may have the data attribute TERMINATED that yields TRUE if the task is *terminated*, and FALSE otherwise.

15.03.10
partie variable
Partie d'un *enregistrement,* composée d'*objets de données*, et dont les *structures de données* ou les *types de données* déclarés correspondants peuvent changer.

NOTE – Le nombre aussi bien que la composition des objets de données peuvent changer.

15.03.11
enregistrement variable
article variable
Enregistrement qui comporte une *partie variable*.

NOTE – L'enregistrement peut comporter des *discriminants* pour indiquer les *types de données* dans la partie variable.

15.03.12
discriminant (nom)
code enregistrement
Élément de langage, similaire à un *paramètre,* qui indique la *structure de données* à utiliser dans un *enregistrement variable* déterminé.

15.03.13
paramètre (en langages de programmation)
Élément de langage destiné à passer des *objets de données* ou des *valeurs de donnée* entre des *modules.*

15.03.14
paramètre effectif
paramètre réel
Paramètre, tel qu'une *expression*, un *identificateur* ou un autre *élément de langage*, utilisé dans un *appel* ou dans une *instanciation générique* pour relier un *objet de données* à la *déclaration* correspondante.

NOTE – La déclaration correspondante est appelée *paramètre formel.*

15.03.15
paramètre formel
paramètre fictif
Paramètre, défini dans la *déclaration* de certains *modules*, et associé à un *paramètre effectif* dans un *appel* ou une *instanciation générique*.

15.03.16
association de paramètres
Association entre les *paramètres formels* et les *paramètres effectifs* correspondants dans un *appel* ou une *instanciation générique.*

15.03.17
attribut de donnée
Caractéristique prédéfinie d'un *type de données,* d'un *objet de données*, d'un *module,* ou de tout autre *élément de langage*.

Exemples: Un *type réel* peut avoir l'attribut de données PRÉCISION avec les *valeurs de donnée* SIMPLE ou DOUBLE. Une tâche peut avoir l'attribut de données TERMINÉ qui prend la valeur VRAI si la tâche est *terminée*, et FAUX dans le cas contraire.

15.03.18
name qualification
qualification
A means of referencing *language constructs* within the *scope* of a portion of a *program* by reference to that portion and an *identifier* declared for the language construct in that portion.

Example: Used for referencing *record* components (B OF A in COBOL), members of a library, language constructs in a *module*.

15.03.19
alias
An alternate *identifier* for a *language construct*.

15.03.20
pointer (in programming languages)
A *data object* whose *data value* is the *address* of another data object.

NOTE – See figure 1.

15.03.21
null pointer
A *pointer* that explicitly does not point to any *data object*.

NOTE – Depending on the *programming language*, the null pointer has a representation called "nil", "null", etc.

15.04 Data types

15.04.01 (17.05.08)
data type
datatype
A defined set of *data objects* of a specified *data structure* and a set of permissible *operations*, such that these data objects act as *operands* in the *execution* of any one of these operations.

Example: An *integer type* has a very simple structure, each occurrence of which, usually called value, is a representation of a member of a specified range of whole numbers and the permissible operations include the usual *arithmetic operations* on these *integers*.

NOTES
1. The term "type" may be used instead of "data type" when there is no ambiguity.
2. See figure 1.

15.04.02
abstract data type
ADT (abbreviation)
A *class* of *data structures* described by a list of *operations* or features available in the data structures and the formal properties of these operations, with the interfaces separated from the internal implementation.

15.04.03
encapsulated type
A *data type* with publicly defined interfaces and privately defined implementation of the internal structure and the associated *operations*.

15.03.18
qualification de donnée
Technique permettant de désigner des *éléments de langage* dans la *portée* d'une partie d'un *programme* en se référant à cette partie et à un *identificateur* déclaré pour l'élément de langage dans cette partie.

Exemple: Cette technique est utilisée pour désigner des composants d'un *enregistrement* (B OF A en COBOL), des membres d'une bibliothèque ou des éléments de langage dans un *module*.

15.03.19
alias
Identificateur de substitution attribué à un *élément de langage*.

15.03.20
pointeur (en langages de programmation)
Objet de données dont la *valeur de donnée* est l'adresse d'un autre objet de données.

NOTE – Voir figure 1.

15.03.21
pointeur nul
Pointeur qui explicitement ne pointe sur aucun *objet de données*.

NOTE – Selon le *langage de programmation*, le pointeur nul a une représentation appelée « nil », « null », etc.

15.04 Types de données

15.04.01 (17.05.08)
type de données
Ensemble déterminé d'*objets de données* d'une *structure de données* spécifiée, accompagné d'un ensemble d'*opérations* permises, de façon que ces objets de données puissent agir en tant qu'*opérandes* lors de l'*exécution* de l'une quelconque de ces opérations.

Exemple: La structure d'un *type entier* est très simple, chacune de ses occurrences, appelée couramment valeur, est une représentation d'un élément d'une certaine portée de nombres entiers et les opérations permises comprennent les *opérations arithmétiques* courantes pour les *entiers relatifs*.

NOTES
1. Le terme « type » peut être utilisé à la place de « type de données » quand il n'y a aucun risque d'ambiguïté.
2. Voir figure 1.

15.04.02
type de données abstrait
Classe de *structures de données* décrite par une liste d'*opérations* ou de caractéristiques disponibles dans cette structure et par les propriétés formelles de ces opérations, et dont les interfaces sont distinctes de la réalisation interne.

15.04.03
type encapsulé
Type de données ayant des interfaces définies de façon publique et une structure interne réalisée de façon privée, ainsi que les *opérations* s'y rapportant.

15.04.04
scalar type
simple type
A *data type*, each instance of which represents a *scalar*.

NOTES
1. Pascal scalar types are either *ordinal types* or *real types*. Ada scalar types are either *discrete types* or real types.
2. See figure 1.

15.04.05
atomic type
A *data type*, each *data object* of which consists of a single nondecomposable *data value*.

15.04.06
logical type
Boolean type
A *data type* whose *data objects* can assume only logical values (usually TRUE or FALSE) and can be operated on only by *Boolean operators*.

NOTE – See also *character type, * enumeration type, * integer type, * real type.*

15.04.07
range (in programming languages)
span (deprecated in this sense)
A contiguous set of *data values* of a *scalar type*.

15.04.08
real type
A *data type*, each *data object* of which represents a *real number*, possibly by approximation.

Example: The decimal number 0.1 in the *binary system* has an infinite number of *digits*.

NOTES
1. Real types are either *fixed-point types* or *floating-point* types.
2. See figure 1.

15.04.09
fixed-point type
implied decimal type
A *real type*, each *data object* of which is expressed in a *fixed-point representation system*.

NOTE – See figure 1.

15.04.10
floating-point type
A *real type*, each *data object* of which is expressed in a *floating-point representation system*.

NOTE – See figure 1.

15.04.04
type scalaire
Type de données dont chaque élément représente un *scalaire*.

NOTES
1. En Pascal et en Ada, les types scalaires sont soit des *types ordinaux*, soit des *types réels*.
2. Voir figure 1.

15.04.05
type atomique
Type de données dont chacun des *objets de données* est constitué d'une *valeur de donnée* non décomposable.

15.04.06
type logique
Type de données dont les *objets de données* peuvent prendre seulement deux valeurs possibles (habituellement VRAI ou FAUX) et ne peuvent être manipulés que par des *opérateurs booléens*.

NOTE – Voir aussi *type caractère*, * *type énumératif*, * *type entier*, * *type réel*.

15.04.07
intervalle (en langages de programmation)
Ensemble contigu de *valeurs de donnée* de *type scalaire*.

15.04.08
type réel
Type de données dont chaque *objet de données* représente un *nombre réel*, parfois par approximation.

Exemple: En *notation binaire* le nombre décimal 0,1 est représenté par un nombre infini de *chiffres*.

NOTES
1. Les types réels sont soit des *types en virgule flottante* soit *des types en virgule fixe*.
2. Voir figure 1.

15.04.09
type en virgule fixe
Type réel dont chaque *objet de données* est représenté en *numération à séparation fixe*.

NOTE – Voir figure 1.

15.04.10
type en virgule flottante
Type réel dont chaque *objet de données* est représenté en *numération à séparation flottante*.

NOTE – Voir figure 1.

15.04.11
ordinal type
discrete type
A *data type*, each *data object* of which represents a member of an ordered countable set.

NOTES
1. Pascal ordinal types are "enumerated", "char", "integer", and "Boolean". Ada ordinal types are either *integer types* or *enumeration types*.
2. See figure 1.

15.04.12
index type
An *ordinal type*, each *data object* of which represents a subscript of an *array*.

15.04.13
integer type
An *ordinal type*, each *data object* of which represents a whole number within a specified *range*.

NOTE – See figure 1.

15.04.14
enumeration type
enumerated type
An *ordinal type* whose *data objects* are explicitly enumerated in the *declaration* of such a *data type*.

NOTE – See figure 1.

15.04.15
numeric type
A *scalar type*, each *data object* of which represents an *integer* or an approximation of a *real number*.

NOTE – See figure 1.

15.04.16
character type
An *ordinal type*, each *data object* of which represents a *character*.

NOTE – See figure 1.

15.04.17
string type
A *data type*, each *data object* of which is a *string*.

NOTE – See figure 1.

15.04.18
pointer type
access type
A *data type*, each *data object* of which is a *pointer*.

NOTE – See figure 1.

15.04.11
type ordinal
Type de données dont chaque *objet de données* représente un membre d'un ensemble comptable ordonné.

NOTES
1. En Pascal, les types ordinaux sont « énumératif », « car », « entier » et « booléen ». En Ada, les « types ordinaux » sont soit des *types entiers*, soit des *types énumératifs*.
2. Voir figure 1.

15.04.12
type index
Type ordinal dont chaque *objet de données* représente un indice dans un *tableau*.

15.04.13
type entier
Type ordinal dont chaque *objet de données* représente un *entier relatif* à l'intérieur d'un *intervalle* déterminé.

NOTE – Voir figure 1.

15.04.14
type énumératif
Type ordinal dont les *objets de données* sont explicitement énumérés dans la *déclaration* de ce *type de données.*

NOTE – Voir figure 1.

15.04.15
type numérique
Type scalaire dont chaque *objet de données* représente un *entier relatif* ou une approximation d'un *nombre réel*.

NOTE – Voir figure 1.

15.04.16
type caractère
Type ordinal dont chaque *objet de données* représente un *caractère*.

NOTE – Voir figure 1.

15.04.17
type chaîne
Type de données dont chaque *objet de données* représente une *chaîne*.

NOTE – Voir figure 1.

15.04.18
type pointeur
Type de données dont chaque *objet de données* est un *pointeur.*

NOTE – Voir figure 1.

15.04.19
array type
A *composite type* whose components are the same *data type*.

NOTES
1. Array types may be organized and referenced as if the components were arranged in columns, rows, etc.
2. See figure 1.

15.04.20
record type
A *composite type* whose components are *field types* or other record types.

Example: A personnel *record* may consist of personal *data* arranged as *fields* or subrecords within this personnel record.

NOTES
1. A record type defines a set of values and *operations*. An instance of such a record type may contain values which themselves are records.
2. See figure 1.
3. This definition is identical to the definition in entry 17.05.13 of ISO/IEC 2382-17. The example and notes have been added.

15.04.21
variant record type
A *record type* that has a *variant part* specifying alternative lists of components.

15.04.22
subtype
A *data type* derived from another data type by one or more *constraints* on that other data type.

15.04.23
base type
host type
underlying type
A *data type* from which a *subtype* is descended.

NOTE – Contrast with *parent type*.

15.04.24
constraint
An adaptation of a *data type* that restricts its *range* or *operations*.

15.04.25
private type
Within a *program*, a *data type* whose structure, set of values, and *operations* are defined but whose availability is restricted to privileged parts of that program.

Example: In Ada, only assignment, equality, and inequality are available to users, except for any operations explicitly made accessible.

15.04.19
type tableau
Type composite dont les composants sont du même *type de données*.

NOTES
1. Les types tableaux peuvent être organisés et référencés comme si leurs composants étaient organisés en colonnes, lignes, etc.
2. Voir figure 1.

15.04.20
type enregistrement
type structure
Type composite dont les *composantes* sont des *types champs* ou d'autres types de données du même genre.

Exemple: Un *enregistrement* de dossier personnel peut contenir des *données* personnelles organisées dans des *champs* ou des sous-enregistrements à l'intérieur de cet enregistrement de dossier personnel.

NOTES
1. Un type enregistrement définit un ensemble de valeurs et d'*opérations*. Un exemple d'un type enregistrement de ce genre peut contenir des valeurs qui sont elles-mêmes des enregistrements.
2. Voir figure 1.
3. Cette définition est identique à la définition de l'article 17.05.13 d'ISO/CEI 2382-17. L'exemple et les notes sont des ajouts.

15.04.21
type union
Type enregistrement ayant une *partie variable* qui spécifie un choix de listes de composants.

15.04.22
sous-type
*Type de donnée*s déduit d'un autre type de données en soumettant ce type de données à une ou plusieurs *contraintes*.

15.04.23
type de base
Type de données dont un *sous-type* est déduit.

NOTE – Comparer à *type parent*.

15.04.24
contrainte
Adaptation d'un *type de données* qui restreint son *intervalle* ou ses *opérations.*

15.04.25
type privé
Dans un *programme*, * *type de données* dont la structure, l'ensemble des *valeurs* et les *opérations* sont définis mais ne sont disponibles que dans certaines parties privilégiées du programme.

Exemple: En Ada, seule l'affectation, l'égalité et l'inégalité sont utilisables en dehors des opérations explicitement rendues disponibles.

15.04.26
limited type
A *private type* for which only explicitly declared *operations* or *data attributes* are available outside a part of a *program* in which it is contained.

15.04.27
parent type
A *data type* that serves as the template for creating new data types.

NOTE – Contrast with *base type*, * *derived type*.

15.04.28
derived type
A *data type* whose *data values* and *operations* are replicas of those of an existing *parent type*.

NOTES
1. *Strong typing* prohibits operations among data values of different derived types, or between a derived type and a parent type, unless explicit *type conversion* is used.
2. The set of data values or the applicable operations of derived types may be reduced or expanded.
3. Contrast with parent type.

15.04.29
type conversion
Transformation of the representation of a *data value* of one *data type* to that of another data type, usually performed to avoid an illegal data type mismatch.

NOTE – Type conversion among *numeric types* frequently is permitted but may cause loss of *accuracy*, *precision*, or both.

15.04.30
strong typing
Enforcement of the requirement that *operands* in a *language construct* must be of *data types* compatible with those of the *operation* or have explicitly undergone *type conversion* before the operation is performed.

Example: The strong typing in Ada makes the addition of 2 + 3.5 illegal, because 2 is an *integer* and 3.5 a *real number*.

15.04.31
weak typing
A relaxation of the rules of *strong typing*.

Example: Weak typing may permit the addition of an *integer* and a floating-point number without explicit *type conversion* of one of the two *operands*.

NOTE – In weak typing, there may or may not be implicit type conversion.

15.04.32
predefined type
A *data type*, referenced by a *predefined identifier*, for which a *programming language* provides appropriate *operations*.

15.04.26
type limité
Type privé dont seules des *opérations* ou des *attributs de données* explicitement déclarés sont disponibles à l'extérieur de la partie de *programme* qui le contient.

15.04.27
type parent
Type de données qui sert de modèle pour déduire de nouveaux types.

NOTE – Comparer avec *type de base*, * *type dérivé*.

15.04.28
type dérivé
Type de données dont les *valeurs* et les *opérations* sont des copies exactes de celles d'un *type parent* existant.

NOTES
1. Le *typage fort* interdit les opérations entre valeurs de types dérivés différents ou entre un type dérivé et un type parent, à moins qu'une *conversion de type* explicite ne soit utilisée.
2. L'ensemble des valeurs de données ou les opérations applicables des types dérivés peuvent être réduits ou élargis.
3. Comparer avec type parent.

15.04.29
conversion de type
Transformation de la représentation d'une valeur d'un *type de données* vers un autre, généralement effectuée pour éviter des conflits de types.

NOTE – La conversion de types entre les *types numériques* est souvent permise mais peut entraîner une perte d'*exactitude* ou de *précision* ou les deux.

15.04.30
typage fort
Obligation faite aux *opérandes* dans un *élément de langage* d'avoir un *type de données* compatible avec l'*opération* effectuée ou de subir une *conversion de type* explicite avant que l'opération ne soit effectuée.

Exemple: En Ada, le typage fort rend l'addition 2 + 3,5 illégale, car « 2 » est un *entier relatif* et « 3,5 » un *nombre réel.*

15.04.31
typage faible
Assouplissement des règles de *typage fort.*

Exemple: Le typage faible permet l'addition d'un *entier relatif* et d'un nombre à virgule flottante sans *conversion de type* explicite de l'un des deux opérandes.

NOTE – En typage faible, il peut y avoir ou non conversion de type implicite.

15.04.32
type prédéfini
Type de données référencé par un *identificateur prédéfini* pour lequel un *langage de programmation* fournit les *opérations* adéquates.

15.04.33
universal type
A *data type* of *numeric literals* and the data type of the result of some *predefined* * *operations* used for compliance with *strong typing*.

Example: In Ada, a number *declaration* (without a data type) assumes a universal type.

15.04.34
anonymous
Pertaining to a *data object* that has no explicit *data type* * *declaration*.

15.04.35
format (in programming languages)
A *language construct* that specifies the representation, in *character* form, of *data objects* in a *record*, *file*, *message*, *storage device*, or *transmission channel*.

15.04.36
picture (in programming languages)
A *language construct* that describes the *format* of *string-type* * *data objects* by means of a model *character literal*.

15.05 Statements and expressions

15.05.01
statement
An explicitly terminated syntactic unit either representing a *declaration* or prescribing a unit of work that includes identification of actions to be performed, *operands* (if any) to be used in performing these actions, and disposition of any *results*.

NOTE – Some *programming languages* do not consider declarations to be statements.

15.05.02
simple statement
elementary statement (deprecated)
A *statement* that encloses no other statement.

15.05.03
compound statement
A *statement* that contains one or more statements, so delimited as to be the syntactic equivalent of a *simple statement*.

15.05.04
assignment statement
assignment
A *simple statement* that replaces the current *data value* of a *variable* with a new data value specified by an *expression*.

15.05.05
exit statement
A *simple statement* used to end the *execution* of an enclosing *language construct*.

15.04.33
type universel
Type de données des libellés numériques et type de données du résultat de certaines *opérations* * *prédéfinies* qui sont utilisés pour se conformer aux règles du *typage fort.*

Exemple: En Ada, une *déclaration* de nombre (sans type de données) implique que le nombre a le type universel.

15.04.34
sans type explicite (qualificatif)
atypique (qualificatif)
Qualifie un *objet de données* dont le *type de données* n'est pas explicitement déclaré**.**

15.04.35
format (en langages de programmation)
Élément de langage qui spécifie la représentation, sous forme de *caractères,* d'*objets de données* dans un *enregistrement,* un *fichier*, un *message*, en *mémoire* ou dans une *voie de transmission*.

15.04.36
image (en langages de programmation)
Élément de langage qui décrit le *format* d'*objets de données* de *type chaîne* au moyen d'un modèle sous forme de *libellé caractère*.

15.05 Instructions et expressions

15.05.01
instruction (1)
énoncé
Unité syntaxique explicitement terminée représentant une *déclaration* ou décrivant une unité de travail qui inclut l'identification des actions à effectuer, les *opérandes* à utiliser, et l'affectation des *résultats* éventuels.

NOTES
1. Certains *langages de programmation* ne considèrent pas les déclarations comme des instructions.
2. Comparer avec *instruction* (2) (07.09.01).

15.05.02
instruction simple
Instruction qui ne contient aucune autre instruction.

15.05.03
instruction composée
Instruction qui contient une ou plusieurs instructions et délimitée de manière à être l'équivalent syntaxique d'une *instruction simple.*

15.05.04
instruction d'affectation
Instruction simple où la *valeur de donnée* d'une *variable* est remplacée par une nouvelle valeur de donnée spécifiée par une *expression*.

15.05.05
instruction de sortie
Instruction simple utilisée pour terminer l'*exécution* d'un *élément de langage* où elle apparaît.

15.05.06
return statement
A *language construct* within a *module* that designates the end of an *execution sequence* (or possibly several such sequences) in that module and causes a *jump* to a specified point in the *calling* module, and possibly provides a *result* to it.

15.05.07
to return (intransitive)
To *execute* a *return statement* that causes a *jump* to the *calling * program*.

15.05.08
to return (transitive)
To provide a *data value* to the *calling * program* when *executing* a *return statement*.

15.05.09
entry
The initiation of an *execution sequence* at the beginning of a *subprogram* or elsewhere as designated by an *entry name* in the subprogram.

15.05.10
entry name
An *identifier* that designates the beginning of an *execution sequence*.

15.05.11
goto statement
A *simple statement* that specifies an explicit transfer of *program* control from its place in the *execution sequence* to a target *statement* that usually is identified by a *label*.

NOTE – The transfer of program control may be equivalent to a *jump*.

15.05.12
unconditional statement
imperative statement
A *statement* that is *executed* without any condition.

15.05.13
conditional statement
A *compound statement* that selects for *execution* one or none of the enclosed *sequences* of *statements* depending on the value of a *conditional expression* of one or more corresponding conditions.

Examples: In Pascal, *if statements* and *case statements* are conditional statements.

15.05.14
conditional expression
An *expression* whose evaluation is used to select subsequent *execution sequences*.

15.05.06
instruction de retour
Élément de langage à l'intérieur d'un *module,* qui désigne la fin d'une ou plusieurs *séquences d'exécution* dans le module et provoque un *saut* vers un point spécifié dans le module *appelant*, et éventuellement lui fournit un *résultat*.

15.05.07
retourner
Exécuter une *instruction de retour* qui provoque un *saut* vers le *programme * appelant*.

15.05.08
renvoyer
rendre
Fournir une *valeur de donnée* au *programme * appelant* lors de l'*exécution* d'une *instruction de retour*.

15.05.09
entrée
Initiation d'une *séquence d'exécution* au début d'un *sous-programme* ou à un autre point désigné par un *nom d'entrée* du sous-programme.

15.05.10
nom d'entrée
Identificateur qui désigne le début d'une *séquence d'exécution*.

15.05.11
instruction GOTO
instruction aller à
Instruction simple qui spécifie explicitement le transfert du déroulement du *programme* depuis son emplacement dans la *séquence d'exécution* jusqu'à une *instruction* cible habituellement identifiée par une *étiquette*.

NOTE – Le transfert de déroulement peut être équivalent à un *saut*.

15.05.12
instruction inconditionnelle
Instruction qui est *exécutée* sans condition.

15.05.13
instruction conditionnelle
Instruction composée qui choisit pour *exécution* l'une ou aucune des *séquences d'instructions* incluses selon la valeur d'une *expression conditionnelle* correspondant à une ou plusieurs conditions.

Exemples: En Pascal, les « *instructions IF* » et les « *instructions CASE* » sont des instructions conditionnelles.

15.05.14
expression conditionnelle
Expression dont la valeur d'évaluation est utilisée pour choisir parmi plusieurs *séquences d'exécution.*

15.05.15
if statement
A *conditional statement* that causes *execution* of the enclosed *sequences* of *statements* or skips them depending on the truth value of the *conditional expression*.

15.05.15
instruction IF
instruction SI
Instruction conditionnelle qui provoque l'*exécution* ou l'omission des *séquences d'instructions* incluses, selon la véracité de l'*expression conditionnelle.*

15.05.16
case statement
A *conditional statement* that selects for *execution* one of a number of alternative *sequences* of *statements* depending on the value of a *conditional expression*.

15.05.16
instruction CASE
instruction de cas
Instruction conditionnelle qui provoque l'*exécution* d'une parmi plusieurs *séquences* d'*instructions*, selon la valeur d'une *expression conditionnelle*.

15.05.17
iteration statement
loop statement
A *compound statement* that includes a mechanism to control repeated *execution* of its enclosed *statements*.

15.05.17
instruction de boucle
instruction en boucle
instruction itérative
Instruction composée qui commande l'*exécution* répétée des *instructions* qu'elle contient.

15.05.18
while-construct
A *language construct* for *iteration* control that defines a test to be performed before each *iteration step*.

15.05.18
élément WHILE
Élément de langage de commande d'*itération*, qui définit un test à effectuer avant chaque *pas d'itération*.

15.05.19
until-construct
A *language construct* for *iteration* control that defines a test to be performed after each *iteration step*.

15.05.19
élément UNTIL
élément jusqu'à
Élément de langage de commande d'*itération,* qui définit un test à effectuer après chaque *pas d'itération*.

15.05.20
for-construct
A *language construct* for *iteration* control that defines the test to be performed for such control, usually based on a *loop-control variable*, and the prescription for the changes of that iteration control variable to be carried out between *iteration steps*.

15.05.20
élément FOR
Élément de langage de commande d'*itération*, qui définit le test à effectuer, habituellement fondé sur une *variable de boucle,* et la description des modifications de ladite variable à effectuer entre chaque *pas d'itération*.

15.05.21
do while statement
repeat while statement
perform while statement
An *iteration statement* where the *iteration* control is incorporated in a *while-construct*.

15.05.21
instruction WHILE
instruction tant que
Instruction de boucle où la commande de l'*itération* fait partie d'un *élément WHILE.*

15.05.22
until statement
repeat until statement
perform until statement
An *iteration statement* where the *iteration* control is incorporated in an *until-construct*.

15.05.22
instruction JUSQU'À
Instruction de boucle où la commande de l'*itération* fait partie d'un *élément JUSQU'À*.

15.05.23
perform statement
A *compound statement* that explicitly specifies transfer of control to one or more COBOL *procedures* and the return of control implicitly whenever *execution* of the specified procedure is completed.

NOTE – The perform statement is also used to control execution of one or more *unconditional statements* which are within its *scope*.

15.05.23
instruction PERFORM
Instruction composée qui spécifie explicitement le transfert de déroulement vers une ou plusieurs *procédures* COBOL et la reprise implicite du déroulement antérieur dès que l'*exécution* de ladite procédure est terminée.

NOTE – L'instruction PERFORM est aussi utilisée pour commander l'exécution d'une ou de plusieurs *instructions inconditionnelles* situées dans la *portée* de cette instruction.

15.05.24
block statement
Any bounded *sequence* of *statements* that can be taken as a single syntactic unit and which may have an *identifier*.

Example: A Pascal *program* can be considered simply as a specific header followed by a block statement, with a *procedure* similarly defined.

NOTES
1. A block statement is a basic syntactic component of *block-structured languages*.
2. In some *programming languages* (e.g., C++), "block" is synonymous with *compound statement*. In other programming languages (e.g., Ada) this concept is given a very specific meaning and may include *declarations* and *exception handlers*.
3. Implementation of a block statement usually has an impact on the *scope* and *lifetime* of *data objects* declared as part of a block statement.

15.05.25
procedure-call statement
procedure call
A *simple statement* that provides the *actual parameters* for and invokes the *execution* of a *procedure*.

NOTE – Contrast with *function call*.

15.05.26
entry-call statement
A *simple statement* that permits a *task* to request a *rendezvous* with another task.

15.05.27
delay statement
A *simple statement* used to suspend *execution* of a *task* that contains a request for a delay.

15.05.28
abort statement
A *simple statement* that causes one or more *tasks* to become abnormal, preventing any further *rendezvous* with such a task.

15.05.29
raise statement
A *simple statement* that propagates an *exception* or causes it to occur.

15.05.30
accept statement
A *compound statement* within a server *task*, that causes this server task to wait for another task or for the *main program* to *execute* an *entry-call statement* for *task synchronization*.

15.05.31
select statement
A *compound statement* that allows a *calling * task* or a called task to choose alternative courses of action or to wait.

15.05.24
instruction bloc
Séquence finie d'*instructions*, qui peut être considérée comme une seule unité syntaxique, et qui peut avoir un *identificateur*.

Exemple: Un *programme* Pascal peut être considéré simplement comme étant un en-tête spécifique suivi par une instruction-bloc accompagné d'une procédure définie de la même façon.

NOTES
1. L'instruction-bloc est la composante syntaxique de base des *langages à structure de blocs*.
2. Dans certains *langages de programmation* (tels que C++), « bloc » est synonyme d'« *instruction composée* ». Dans d'autres langages de programmation (tels qu'Ada) le concept est très spécifique et peut inclure des *déclarations* et des *processeurs d'exception*.
3. L'utilisation d'une instruction-bloc a généralement des répercussions sur *la portée* et la *durée de vie* des *objets de données* déclarés à l'intérieur du bloc.

15.05.25
instruction d'appel de procédure
Instruction simple qui fournit les *paramètres effectifs* d'une procédure et en provoque l'*exécution*.

NOTE – Comparer avec *appel de fonction*.

15.05.26
instruction point de convergence
Instruction simple qui permet à une *tâche* de demander un *rendez-vous* avec une autre tâche.

15.05.27
instruction d'attente
Instruction simple utilisée pour suspendre l'*exécution* de la *tâche* qui la contient.

15.05.28
instruction de sabordage
Instruction simple qui rend une ou plusieurs *tâches* anormales.

15.05.29
instruction de déclenchement d'exception
Instruction simple qui provoque ou propage une *exception*.

15.05.30
instruction d'attente de réception
Instruction composée, dans une *tâche* serveuse, qui fait que cette tâche serveuse va attendre l'exécution d'une *instruction point de convergence* par une autre tâche ou par le *programme principal*, à des fins de *synchronisation des tâches*.

15.05.31
instruction de choix
instruction de sélection
Instruction composée qui permet à une *tâche* appelante ou appelée de choisir parmi plusieurs *séquences d'exécution* ou d'attendre.

15.05.32
selective-wait statement
A *select statement* that waits for a *call* from an *entry-call statement* before it *executes* its *sequence* of *statements*.

15.05.33
expression
A *language construct* that defines the computation of a *data value* as a *result* from one or more *operands*.

NOTE – Operands may be literals, *identifiers*, * *function calls*.

15.05.34
mixed mode (adj.)
mixed type (adj.)
Pertaining to an *expression* that contains two or more different *data types*.

15.05.35
Boolean expression
A *language construct* that defines the computation of a logical value.

15.05.36
operator precedence
An ordering rule defining the sequence of the application of *operators* within an *expression*.

NOTE – The ordering rule may specify the evaluation direction.

15.06 Parts of programs

15.06.01
module
program unit
A part of a *program* developed to be discrete or identifiable with respect to actions such as *compilation*, *binding*, or *execution*, and that may interact with other programs or parts of programs.

NOTES
1. The concept referred to by the term “module” may vary according to the different *programming languages*.
2. See figure 2.

15.06.02
body (in programming languages)
A *language construct* that comprises the executable part of a *statement* or *module*.

15.06.03
subprogram
A *module* that has an *identifier* and that is invoked into the *control flow* from another *program* or by another module by means of a specific *language construct* and from which the control flow *returns* to the invoking program or module.

15.05.32
instruction d'attente sélective
Instruction de choix qui attend un *appel* provenant d'une *instruction point de convergence* avant d'*exécuter* sa propre *séquence* d'*instructions*.

15.05.33
expression
Élément de langage qui définit le calcul d'une *valeur de donnée* comme le *résultat* d'un ou plusieurs *opérandes*.

NOTE – Les opérandes peuvent être des libellés, des *identificateurs*, des *appels de fonction*.

15.05.34
de type mixte (qualificatif)
Qualifie une *expression* qui contient plusieurs *types de données* différents.

15.05.35
expression booléenne
Élément de langage qui définit le calcul d’une valeur logique.

15.05.36
priorité des opérateurs
préséance des opérateurs
Règle qui définit l'ordre d'application des *opérateurs* dans une *expression*.

NOTE – La règle peut spécifier si l'évaluation est faite de gauche à droite ou de droite à gauche.

15.06 Éléments de programme

15.06.01
module
unité de programme
Partie de *programme* conçue pour être distincte ou identifiable dans le cadre d'actions telles que la *compilation*, l'*édition de liens*, ou l'*exécution*, et qui peut interagir avec d'autres programmes ou parties de programmes.

NOTES
1. La notion désignée par le terme « module » peut varier selon les *langages de programmation*.
2. Voir figure 2.

15.06.02
corps (en langages de programmation)
Élément de langage qui constitue la partie exécutable d'une *instruction* ou d'un *module*.

15.06.03
sous-programme
Module ayant un *identificateur* et qui est *appelé* par un *élément de langage* dans le *flux de commande* d'un autre *programme* ou d’un autre module et d’où le flux de commande reviendra ultérieurement au programme ou module appelant.

15.06.04
coroutine
A *subprogram* that, when *called* again after an *execution*, resumes at the location to which its previous execution *returned*.

15.06.05
call (in programming languages)
The *instruction* to transfer control from one *module* to another, usually with the implication that control will be given back to the *calling* module.

NOTE – A call usually specifies *parameters* to be passed to and from the called module.

15.06.06
to call
To *execute* a *call*.

15.06.07
call by name
A *call* in which the *calling * module* provides to the called module the names of one or more *parameters* to be evaluated each time the associated parameter is used in the called module.

15.06.08
call by reference
call by address
call by location
A *call* in which the *calling * module* provides to the called module the *addresses* of the *parameters* to be passed.

NOTE – In a call by reference, the called module has the ability to change the values of the parameters *stored* by the calling module.

15.06.09
call by value
A *call* in which the *calling * module* provides to the called module the actual values of the *parameters* to be passed.

NOTE – In a call by value, the called module cannot change the values of the parameters *stored* by or for the calling module.

15.06.10
subprogram call
A *call* that invokes a *subprogram*.

Examples: *Procedure-call statement*, * *function call*.

15.06.11
procedure
subroutine
A *subprogram* that does not *return* a *data value*, except as part of the *parameter* mechanism.

NOTES
1. In COBOL, a procedure is a paragraph, or a group of logically successive paragraphs, or a section (consisting of zero or more paragraphs) within the procedure division.
2. In some *programming languages* (i.e., C and C++), the procedure *language construct* is not differentiated from the *function* language construct except that returned *data values* may be void or not used.

15.06.04
coroutine
Sous-programme qui, quand il est *appelé* à nouveau après une *exécution*, reprend à l'emplacement de l'*instruction de retour* de son exécution précédente.

15.06.05
appel (en langages de programmation)
Instruction destinée à transférer la commande d'un *module* à un autre, et qui implique généralement le retour de la commande au module *appelant*.

NOTE – Un appel spécifie généralement les *paramètres* échangés entre les modules appelant et appelé.

15.06.06
appeler
Exécuter un *appel*.

15.06.07
appel par nom
Appel dans lequel le *module * appelant* fournit au module appelé les noms d'un ou plusieurs *paramètres* qui doivent être évalués chaque fois que le paramètre associé est utilisé dans le module appelé.

15.06.08
appel par référence
appel par adresse
Appel dans lequel le *module * appelant* fournit au module appelé les *adresses* des *paramètres*.

NOTE – Dans un appel par référence, le module appelé peut changer la valeur des paramètres *stockés* par le module appelant.

15.06.09
appel par valeur
Appel dans lequel le *module * appelant* fournit au module appelé les valeurs réelles des *paramètres*.

NOTE – Dans cet appel, le module appelé ne peut pas changer les valeurs des paramètres *stockés* par ou pour le module appelant.

15.06.10
appel de sous-programme
Appel d'un *sous-programme.*

Exemples: *Appel de procédure*, * *appel de fonction*.

15.06.11
procédure
Sous-programme qui ne *renvoie* pas de *valeur de donnée*, sauf en tant que partie d'un *paramètre*.

NOTES
1. En COBOL, une procédure est un paragraphe ou un groupe de paragraphes qui se suivent logiquement, ou bien une section comprenant zéro ou un ou plusieurs paragraphes, à l'intérieur de la division procédure.
2. Dans certains *langages de programmation*, par exemple C et C++, l'*élément de langage* procédure n'est pas distinct de l'élément de langage *fonction* sauf que les *valeurs de données* renvoyées peuvent être vides ou ne pas être utilisées.

15.06.12
function (in programming languages)
A *subprogram*, usually with *formal parameters*, that produces a *data value* which it *returns* to the place of the invocation.

NOTE – A function may also produce other changes through the use of parameters.

15.06.13
function call
A *language construct* that provides the *actual parameters* for the invocation of the *execution* of a *function* and causes the execution.

NOTES
1. A function call may be used as an *operand* in an *expression* or as an actual parameter of a *subprogram call*.
2. Contrast with *procedure-call statement*.

15.06.14
transaction call
A *function call* that permits a *task* to request a *rendezvous* with another task.

15.06.15
subunit
The separately *compiled* * *body* of a *module*.

15.06.16
body stub
A form of a *body* that indicates that the executable part of a *module* is defined in a *subunit*.

15.06.17
connection (in programming languages)
A technique that enables interaction among *modules*, particularly *procedure-call statements* to *asynchronous* * *procedures*.

15.06.18
named parameter association
assignment by name
In a *subprogram call*, the explicit naming of the *formal parameters* corresponding to *actual parameters* to establish *parameter association*.

NOTES
1. In named parameter association, actual parameters can be given in any order.
2. Contrast with *positional parameter association*.

15.06.19
positional parameter association
In a *subprogram call*, the correspondence of an *actual parameter* with a *formal parameter* in the same position in the *declaration* of the *subprogram*.

NOTE – Contrast with *named parameter association*.

15.06.12
fonction (en langages de programmation)
Sous-programme, généralement muni de *paramètres formels*, qui produit une *valeur de donné* qu'il *renvoie* à l'endroit où il a été appelé.

NOTE – Une fonction peut également produire d'autres changements par l'utilisation de paramètres.

15.06.13
appel de fonction
Élément de langage qui fournit les *paramètres effectifs* d'une *fonction* et en provoque l'*exécution*.

NOTES
1. Un appel de fonction peut être utilisé comme *opérande* dans une *expression* ou comme paramètre effectif dans un *appel de sous-programme*.
2. Comparer avec *instruction d'appel de procédure*.

15.06.14
appel de transaction
Appel de fonction qui permet à une *tâche* de demander un *rendez-vous* avec une autre tâche.

15.06.15
sous-unité
Corps d'un *module*, * *compilé* séparément.

15.06.16
corps souche
Forme de *corps* qui indique que la partie exécutable d'un *module* est définie dans une *sous-unité*.

15.06.17
connexion (en langages de programmation)
Technique permettant l'interaction entre *modules*, en particulier entre les *instructions* d'*appel de procédure* et les *procédures* * *asynchrones*.

15.06.18
affectation par nom
Dans un *appel de sous-programme*, désignation explicite des *paramètres formels* correspondant aux *paramètres effectifs* pour établir l'*association de paramètres*.

NOTES
1. Dans l'affectation par nom, les paramètres effectifs peuvent être spécifiés dans n'importe quel ordre.
2. Comparer avec *affectation par position*.

15.06.19
affectation par position
affectation par paramètre positionnel
affectation par paramètre à position fixe
Dans un *appel de sous-programme*, correspondance établie entre un *paramètre effectif* et un *paramètre formel* ayant la même position dans la *déclaration* du *sous-programme*.

NOTE – Comparer avec *affectation par nom*.

15.06.20
formal parameter mode
A characteristic that indicates whether a *formal parameter* may be evaluated without changing it, may be given a new value, or may be evaluated and changed.

15.06.21
macroinstruction
macro
An *instruction* that invokes a *macrodefinition* at the level of the *calling * programming language*.

15.06.22
macrocall
A *statement* that invokes a *macrodefinition* at the level of the *calling * programming language*.

15.06.23
macrodefinition
A predefined *sequence* of commands, *statements*, or *instructions* that replaces each corresponding invoking *macroinstruction* or *macrocall.*

15.06.24
package (in programming languages)
A *module* designed to provide abstraction, *encapsulation*, or *information hiding* through grouping of logically related *language constructs*, such as *data types*, *data objects* of these data types, and *subprograms* with *parameters* of these data types.

15.06.25
package declaration
The separate *declaration* of those *language constructs* whose specifications are required outside the *package*, either for interfacing or for *compilation* purposes.

15.06.26
visible part
That part of a *package declaration* that provides details required by users of *objects* or services of the *package*.

15.06.27
private part
That part of a *package declaration* that provides structural details needed by the development process but irrelevant and unaccessible to the functional users of the *package*.

15.06.28
generic
Pertaining to a *language construct* that serves as a template for creating an actual language construct for applicable *data types* in compliance with the rules of *strong typing*.

15.06.29
generic declaration
The *declaration* of a *generic * language construct* that introduces the generic *parameters* which are to be replaced by *actual parameters* during a *generic instantiation*.

15.06.20
mode de paramètre formel
Caractéristique qui indique si un *paramètre formel* peut être évalué sans modification, ou recevoir une nouvelle valeur, ou être à la fois évalué et modifié.

15.06.21
macro-instruction
macroinstruction
macro
Instruction qui appelle une *macro-définition* au niveau du *langage de programmation* appelant.

15.06.22
appel de macro-instruction
appel de macro
Instruction qui appelle une *macro-définition* au niveau du *langage de programmation* appelant.

15.06.23
macrodéfinition
Séquence prédéfinie de commandes ou *d'instructions* qui remplace la *macro-instruction* correspondante ou l'*appel de macro-instruction* correspondant.

15.06.24
paquetage (en langages de programmation)
Module conçu pour fournir l'abstraction, l'*encapsulation*, ou le *masquage d'information* par le groupement d'*éléments de langage* logiquement associés, tels que les *types de données*, les *objets de données* ayant ces types, et les *sous-programmes* dont les *paramètres* ont ces types.

15.06.25
déclaration de paquetage
Déclaration séparée des *éléments de langage* dont les spécifications sont nécessaires à l'extérieur d'un *paquetage*, soit pour l'interfaçage, soit pour les besoins de la *compilation*.

15.06.26
élément visible
partie visible
Partie d'une *déclaration de paquetage* qui fournit les détails nécessaires aux utilisateurs des *objets* ou services définis dans le *paquetage*.

15.06.27
élément privé
partie privée
Partie d'une *déclaration de paquetage* qui fournit les détails structuraux nécessaires au processus de développement, mais qui sont superflus et invisibles pour les utilisateurs des fonctionnalités du *paquetage*.

15.06.28
générique
Qualifie un *élément de langage* qui sert de modèle pour créer un élément de langage effectif pour des *types de données* utilisables selon les règles du *typage fort.*

15.06.29
déclaration générique
Déclaration d'un *élément de langage* **générique* qui introduit les *paramètres* génériques qui doivent être remplacés par des *paramètres effectifs* lors d'une *instanciation générique*.

15.06.30
generic body
The *body* of a *generic * language construct* that serves as a template for the bodies of corresponding actual language constructs during a *generic instantiation*.

15.06.31
generic operation
An *operation* that is *overloaded* and does not designate one specific operation but rather provides *formal parameters* for *actual parameters* of specific *data types*.

Example: The *lexical token* "+" can mean *integer* addition, real addition, set union, concatenation, etc.

15.06.32
generic package
A *package* designed to provide templates for related *algorithms* or *operations*.

Examples: Generic packages for trigonometric functions, *stack * operations*, financial functions, etc.

15.06.33
generic module
A parameterized template for creating *modules* by *generic instantiation*.

NOTE – The *parameters* of the template are of a *generic* nature and should not be confused with *formal parameters* of the resulting modules.

15.06.34
generic instantiation
The process of resolving *generic * parameters* from a *generic module* in order to create a concrete *module*.

15.06.35
generic instance
A concrete *module* created from a *generic module* by *generic instantiation*.

15.07 Tasks

15.07.01
main program
The first *module* of a *program* to be *executed* and that may invoke the *execution* of other modules.

15.07.02
task (in programming languages)
A *module* that can be *executed * concurrently* with other modules either on a *multiprocessor* or *interleaved* on one *processor*.

NOTE – The distinction between a task and a module from the point of view of *execution* control is not always precise.

15.07.03
critical section
A portion of a *task* during the *execution* of which other parts of this or other tasks are prohibited from *execution*.

15.06.30
corps générique
Corps d'un *élément de langage *générique* qui sert de modèle pour les corps des éléments de langage effectifs correspondants lors d'une *instanciation générique*.

15.06.31
opération générique
Opération qui est *surchargée* et ne désigne pas une opération spécifique mais fournit des *paramètres formels* pour des *paramètres effectifs* ayant des *types de données* spécifiques.

Exemple: L'*unité lexicale* « + » peut signifier une addition d'*entiers relatifs*, une addition de *nombres réels*, une union d'ensembles, une concaténation, etc.

15.06.32
paquetage générique
Paquetage conçu pour fournir des modèles pour des *algorithmes* ou des *opérations* s'y rapportant.

Exemples: Paquetages génériques pour des fonctions trigonométriques, des *opérations de bits*, des fonctions financières, etc.

15.06.33
module générique
Modèle paramétré permettant de créer des *modules* par *instanciation générique*.

NOTE – Les paramètres du modèle sont *génériques* par nature et ne doivent pas être confondus avec les *paramètres formels* des modules produits.

15.06.34
instanciation générique
Traitement qui assigne les *paramètres * génériques* d'un *module générique* pour produire un *module* réel.

15.06.35
instance générique
Module réel produit par l'*instanciation générique* d'un *module générique*.

15.07 Tâches

15.07.01
programme principal
Premier *module* d'un *programme* à exécuter et qui peut appeler l'*exécution* d'autres modules.

15.07.02
tâche (en langages de programmation)
Module qui peut être *exécuté* de façon *concurrente* avec d'autres modules soit par un *multiprocesseur*, soit *imbriqués* sur un seul *processeur*.

NOTE – La distinction entre une tâche et un module du point de vue de la commande d'*exécution* n'est pas toujours précise.

15.07.03
section critique
Partie d'une *tâche* pendant l'exécution de laquelle les autres parties de cette tâche ou d'autres tâches ne doivent pas être exécutées.

15.07.04
task synchronization
The means by which *tasks* coordinate their activities in time.

Examples: *Semaphore*, * *monitor*, * *rendezvous*.

15.07.05
rendezvous
The interaction between two *tasks* which is time-coordinated at a certain point in each *process* of task *execution* and where one process may wait for the other.

15.07.06
semaphore
A *data structure* for controlling, by means of a *queue*, *access* to the *resources* that are available to more than one *task*, but only to one task at a time.

15.07.07
monitor (in programming languages)
A shared *data object* together with a set of *operations* that may manipulate the data object in order to control requests for *resources* or *access* to the resources that are available to *parallel * processes*, but only to one process at a time.

15.08 Execution

15.08.01
execution sequence
The order of the elaboration of *declarations* and of the *execution* of *statements* and parts of statements.

15.08.02
control flow
A path the *execution sequence* may take through a *program*.

NOTE – An abstraction of all the control flows can be represented by a *control-flow diagram*.

15.08.03
side effect
Any indirect consequence caused by the *execution* of an *expression*, *statement*, or *subprogram*.

NOTE – Side effects may be intended, for example, to change the *data value* of a *parameter* passed by a *function*.

15.09 Object-oriented programming

15.09.01
information hiding
The principle of denying *access* to or knowledge of a *language construct* or specific details thereof, except for those details considered essential for the user to know.

15.09.02
to encapsulate
To apply *information hiding* to a *language construct*.

15.07.04
synchronisation des tâches
Moyen de coordination temporelle des activités des *tâches.*

Exemples: *Sémaphore*, * *moniteur*, * *rendez-vous*.

15.07.05
rendez-vous
Interaction de deux *tâches,* synchronisée à un point donné dans chaque *processus* d'*exécution* des tâches et où un processus peut attendre l'autre.

15.07.06
sémaphore
Structure de données servant à contrôler, au moyen d'une *file d'attente*, l'*accès* aux *ressources* qui sont disponibles pour plusieurs *tâches*, mais pour une seule à la fois.

15.07.07
moniteur (en langages de programmation)
Objet de données commun à un ensemble d'*opérations* pouvant manipuler l'objet de données afin d'organiser les demandes de *ressources* ou d'accès aux ressources qui sont disponibles pour des *processus * parallèles*, mais seulement un à la fois.

15.08 Exécution

15.08.01
séquence d'exécution
Ordre dans lequel les *déclarations* sont élaborées et les *instructions* et parties d'instructions sont exécutées.

15.08.02
flux de commande
Cheminement que peut suivre la *séquence* d'*exécution* d'un *programme.*

NOTE – L'ensemble des *flux de commande* peut être représenté graphiquement.

15.08.03
effet secondaire
Toute conséquence indirecte provoquée par l'*exécution* d'une *expression*, d'une *instruction* ou d'un *sous-programme.*

NOTE – Un effet secondaire peut être voulu, par exemple, pour changer la *valeur de données* d'un *paramètre* dans une *fonction*.

15.09 Programmation orientée objet

15.09.01
masquage d'information
Principe qui consiste à refuser l'*accès* à un *élément de langage* ou à certains de ces détails, ou à en prendre connaissance, à l'exception des détails considérés comme essentiels pour l'utilisateur.

15.09.02
encapsuler
Appliquer le *masquage d'information* à un *élément de langage*.

15.09.03
encapsulation
The process or the result of *encapsulating*.

15.09.04
private
Pertaining to characteristics of *language constructs* that are not directly available to the user of these language constructs.

15.09.05
object (in programming languages)
A set of *operations* and *data* that *store* and retain the effect of the operations.

NOTE – Objects are implemented as *packages* or *tasks* in Ada, as "modules" in Modula-2, and as "objects" in Smalltalk.

15.09.06
message (in programming languages)
A request for an *object* to perform one of its *operations*.

15.09.07
protocol (in programming languages)
The set of rules that determines the behavior of *objects* in the exchange of *messages*.

15.09.08
method (in programming languages)
An *operation* that an *object * executes* upon receipt of a *message*.

15.09.09
class (in programming languages)
A template for *objects* that defines the internal structure and the set of *operations* for *instances* of such objects.

NOTE – In *object-oriented * programming*, classes are comparable to *data types* in some *programming languages*, such as C and Pascal.

15.09.10
polymorphism
The ability of different *objects* to respond to the same *message* differently.

15.09.11
inheritance
The copying of all or part of the internal structure and of the set of *operations* from one *class* to a subordinate class.

15.09.12
delegation
A means that permits an *object* to assign servicing of a *message* to another object.

15.09.03
encapsulage
encapsulation
Action d'*encapsuler* ou résultat de cette action.

15.09.04
privé
Qualifie les caractéristiques d'*éléments de langage* qui ne sont pas directement disponibles à leur utilisateur.

15.09.05
objet (en langages de programmation)
Ensemble d'*opérations* et de *données* qui *mémorisent* l'effet des opérations.

NOTE – Les objets sont mis en œuvre comme des *paquetages* ou des *tâches* en Ada, comme des « modules » en Modula-2, et comme des « objets » en Smalltalk.

15.09.06
message (en langages de programmation)
Demande qu'un *objet* effectue une de ses *opérations.*

15.09.07
protocole (en langages de programmation)
Ensemble de règles qui détermine le comportement d'*objets* dans l'échange de *messages.*

15.09.08
méthode (en langages de programmation)
Opération qu'un *objet * exécute* à la réception d'un *message.*

15.09.09
classe (en langages de programmation)
Modèle pour les *objets* qui définit la structure interne et l'ensemble des *opérations* pour les *occurrences* de ces objets.

NOTE – En *programmation * orientée objet*, les classes se comparent aux *types de données* de certains *langages de programmation*, tels que C et Pascal.

15.09.10
polymorphisme
Capacité de différents *objets* de répondre au même *message* de façon différente.

15.09.11
héritage
Reproduction de tout ou partie de la structure interne et de l'ensemble des *opérations* d'une *classe* à une classe subordonnée.

15.09.12
délégation
Moyen de permettre à un *objet* d'attribuer le service d'un *message* à un autre objet.

15.09.13
object-oriented
Pertaining to a technique or a *programming language* that supports *objects*, *classes*, and *inheritance*.

NOTE – Some authorities list the following requirements for object-oriented *programming*: *information hiding* or *encapsulation*, *data* abstraction, *message* passing, *polymorphism*, *dynamic binding*, and inheritance.

15.10 Features and characteristics

15.10.01
subscripting
Referencing an *array* element by means of an array reference and one or more *expressions* that, when evaluated, denote the position of the element.

15.10.02
indirect referencing
Referencing via a *data object* that points to a referenced *language construct*.

NOTE – The referencing may be done along a chain of data objects, in which case each data object, except the last, points to the next, the last data object pointing to the referenced language construct.

15.10.03
to initialize
To give a *data value* to a *data object* at the beginning of its *lifetime*.

15.10.04
dynamic storage allocation
Allocation of *storage* space to *data objects* only for the duration of the *execution* of their *scope*.

15.10.05
extensibility
The capability of a *programming language* to allow the specification of new *language constructs* and their use in the same syntactic manner as the standard language constructs.

15.09.13
orienté objet
par objets
Qualifie une technique ou un *langage de programmation* qui prend en charge les *objets*, les *classes*, et permet l'*héritage*.

NOTE – Certains spécialistes donnent les exigences suivantes pour la programmation orientée objet: *masquage d'information* ou *encapsulage*, abstraction des *données*, passation de *messages*, *polymorphisme*, *liaison dynamique* et héritage.

15.10 Fonctions et caractéristiques

15.10.01
indiçage
désignation
Procédé permettant d'obtenir la référence d'un élément d'un *tableau* au moyen de la référence du tableau et d'une ou plusieurs *expressions* dont l'évaluation fournit la position de cet élément dans le tableau.

15.10.02
référence indirecte
Procédé permettant de désigner un *élément de langage* au moyen d'un *objet de données* qui pointe vers cet élément.

NOTE – La référence peut être obtenue au moyen d'une chaîne d'objets de données dans laquelle chaque objet de données pointe vers l'objet suivant, sauf le dernier qui pointe vers l'élément de langage désiré.

15.10.03
initialiser
Fournir une *valeur de donnée* à un *objet de données*, au début de sa *durée de vie.*

15.10.04
affectation dynamique de mémoire
Affectation temporaire d'espace en *mémoire* à des *objets de données*, seulement pour la durée d'*exécution* correspondant à leur *portée*.

15.10.05
extensibilité
Capacité d'un *langage de programmation* de permettre la spécification de nouveaux *éléments de langage* et leur utilisation de la même manière syntaxique que les éléments de langage normaux.

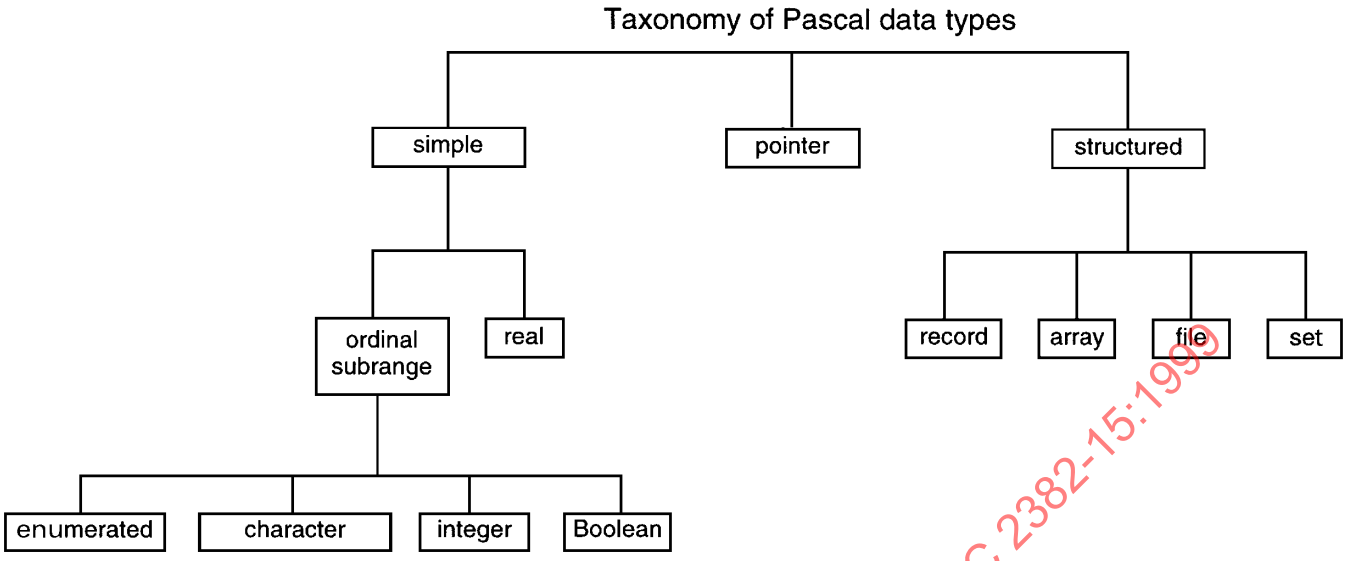

Figure 1 - Examples of data types in Ada and Pascal

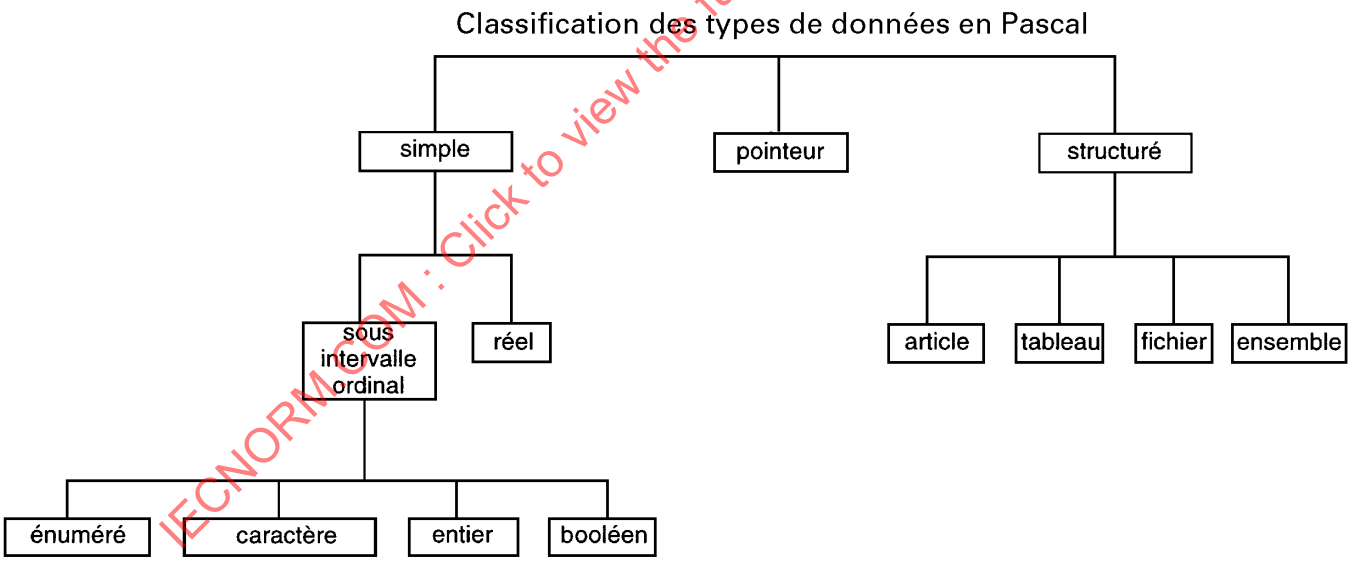

Figure 1 - Exemples de types de données en Ada et en Pascal

Taxonomy of Ada data types

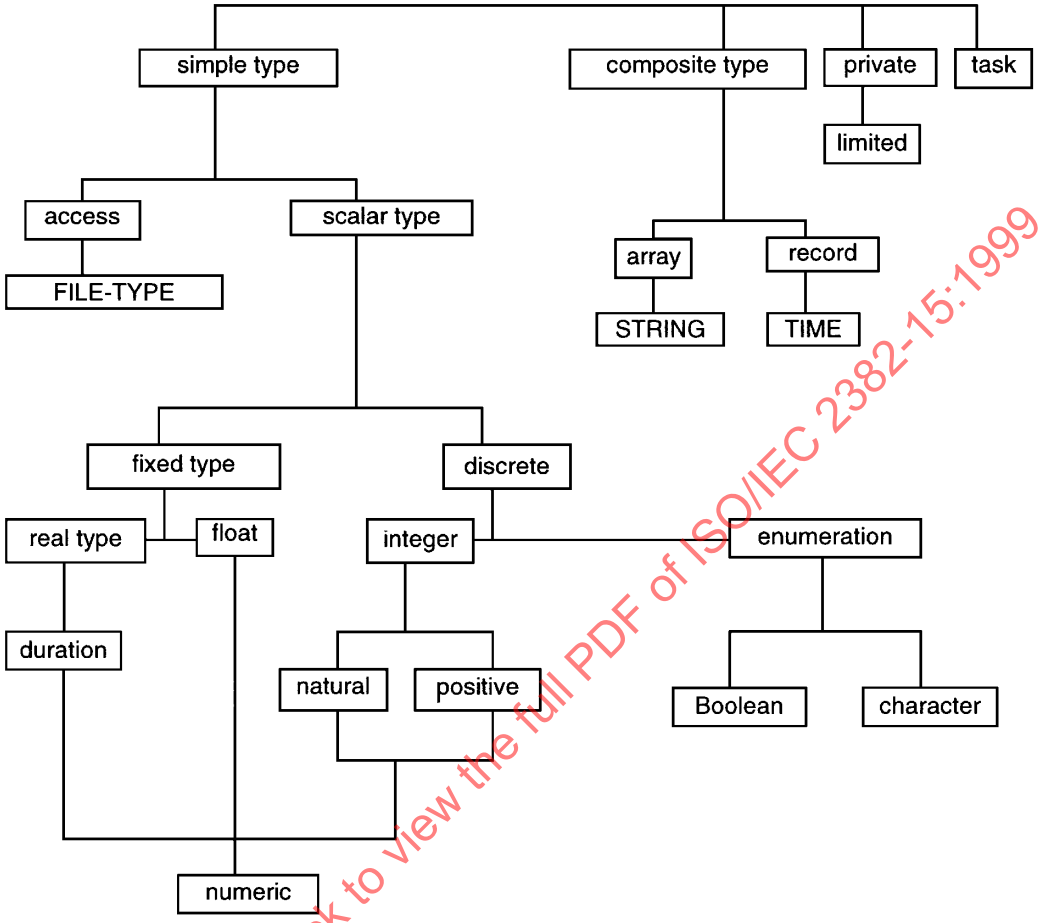

Figure 1 - Examples of data types (continued)

Legend : CAPITAL LETTERS indicate predefined identifiers.
All lower case indicates reserved words in ADA

Classification des types de données en Ada

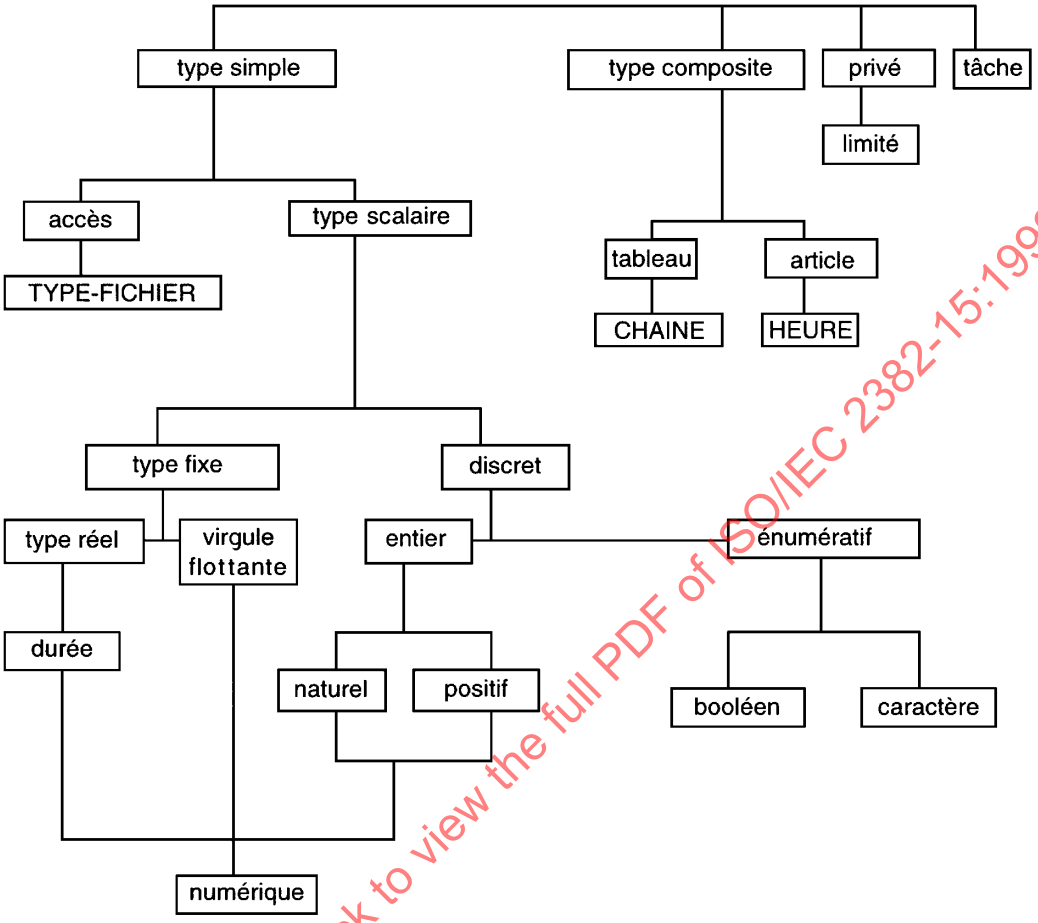

Figure 1 - Exemples de types de données (suite)

Légende : Les MAJUSCULES indiquent des identificateurs pré-déterminés.
Les minuscules indiquent des mots réservés en Ada.

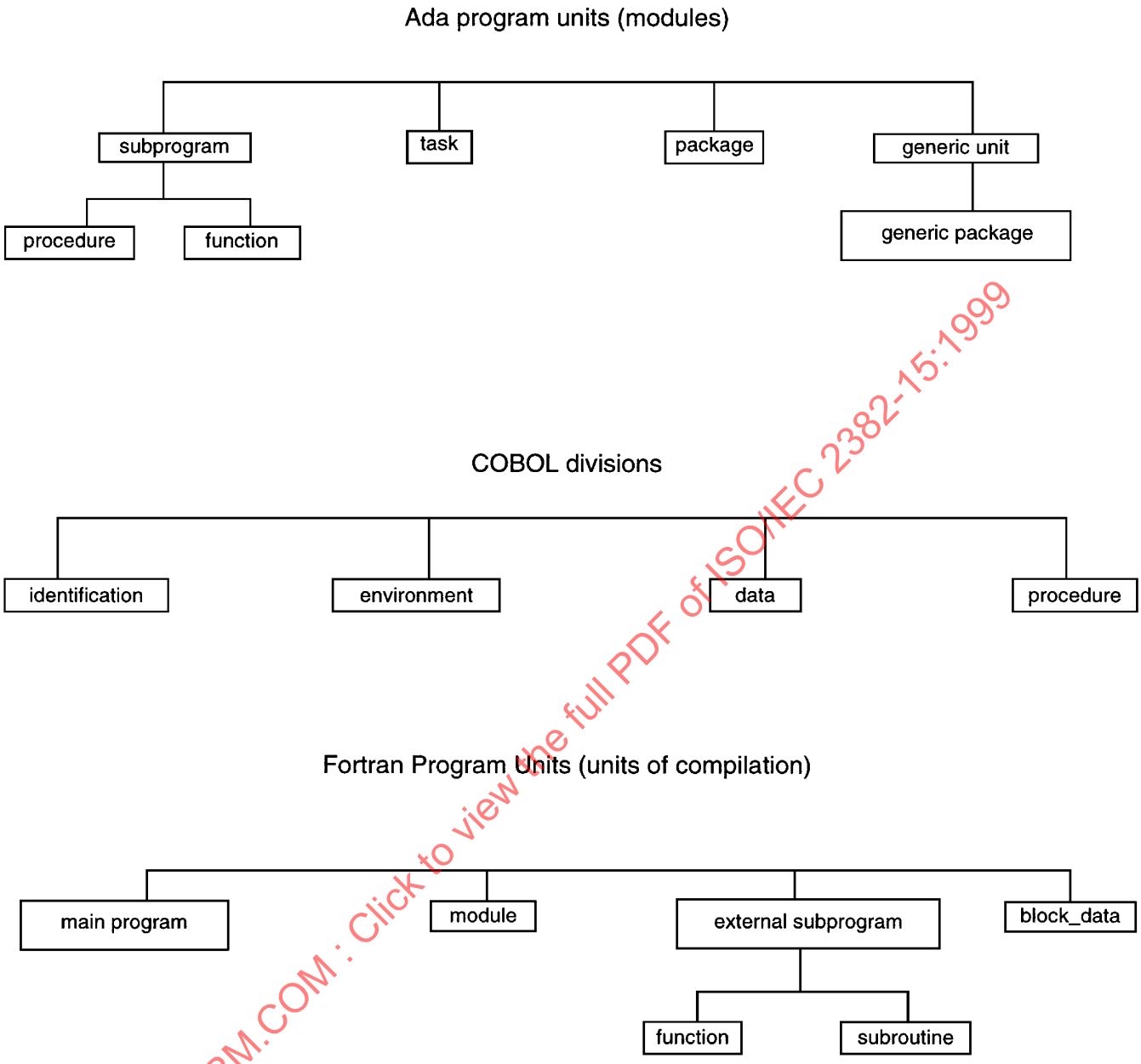

Figure 2 - Examples of parts of programs

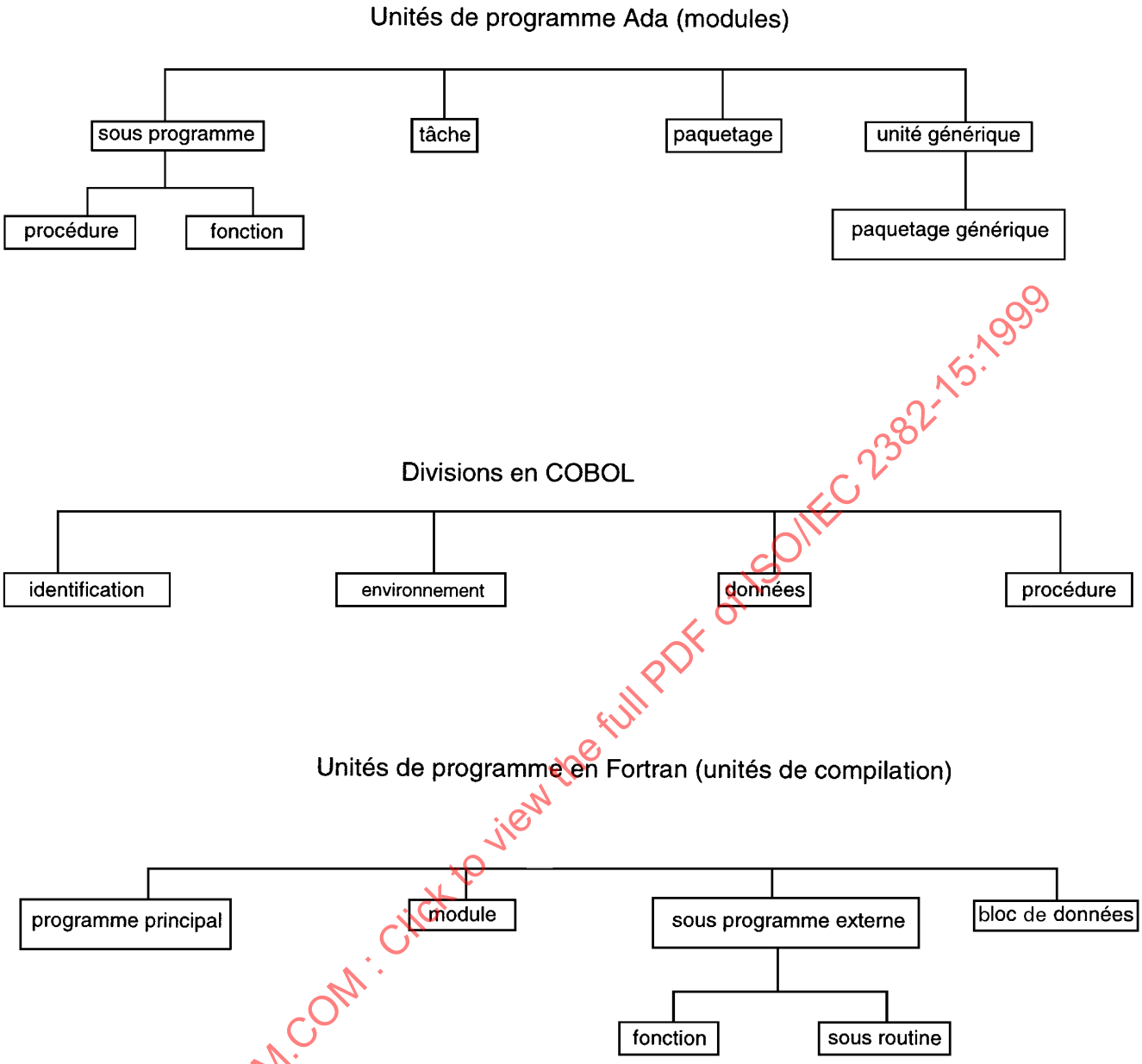

Figure 2 - Exemples de parties de programmes

English Alphabetical Index

A

B

C

D

E

F