Навчально-методична розробка до курсу «Англійська мова» для спеціальності «Комп’ютерна інженерія» ч.2 Ужгород − 2023 2 УДК 811.111(076):004 A64 Англійська мова для студентів спеціальності «Комп’ютерна інженерія» (English for ……): Навчально-методична розробка до курсу «Англійська мова» для спеціальності «Комп’ютерна інженерія» / Уклад. Н.І.Годованець, В.П.Леган, І.О. Бура. Ужгород: Видавництво УжНУ «Говерла», - Ч-2, 2023. 76 с. Рецензенти: кандидат філологічних наук, доцент Герцовська Н. О. кандидат педагогічних наук, доцент Канюк О.Л. Мета навчально-методичного посібника − забезпечити практичне оволодіння студентами лексичними та мовленнєвими моделями, необхідними для спілкування англійською мовою за фахом. Рекомендовано до друку Вченою радою факультету іноземної філології ДВНЗ «УжНУ» від 16 лютого 2023 року, протокол № 1. 3 ПЕРЕДМОВА З прискоренням глобалізації і поширенням міжнародних ділових зв’язків України з іншими державами на політичному та економічному рівнях зростає потреба у висококваліфікованих фахівцях, які здатні вільно володіти основами ділового іншомовного спілкування. У таких умовах важливим є усвідомлення майбутніми фахівцями різних галузей необхідності їх майбутніх зв’язків із міжнародним середовищем, а одним з першочергових завдань освіти стає якісна підготовка фахівців, здатних до успішної професійної діяльності в межах світової спільноти. В цьому контексті істотно змінюються вимоги до володіння іноземною мовою фахівцями всіх рівнів, першорядного значення набувають практичні навички, що передбачають знання ділової іноземної мови в усному та писемному мовленні, вміння використовувати іноземну мову у своїй професійній діяльності. Пропонований посібник має на меті ознайомити з основами іншомовного ділового спілкування, розширити словниковий запас за рахунок спеціальної лексики, виробити комунікативні навички ділового мовлення та навички ділового листування, організації ділових зустрічей та переговорів, оволодіння мовним матеріалом, необхідним під час здійснення ділових подорожей. Посібник складається з тематичних розділів, кожен з яких містить 1) тексти інформативного характеру, які допоможуть орієнтуватися в певних ситуаціях професійної сфери, 2) лексичний матеріал з найчастіше вживаними мовленнєвими конструкціями відповідної тематики, та 3) систему вправ для успішного засвоєння та вдосконалення комунікативних навичок професійного спілкування. 4 C O N T E N T S TOPIC 1 Programming Languages ……………………...................... 5 TOPIC 2. Imperative Mood.………………………………………… 18 TOPIC 3. Computer Networks.………………………………………. 19 TOPIC 4. Gerund.…………………………………………………… 24 TOPIC 5. Computer software........................……………………….. 28 TOPIC 6. The Infinitive.…………………………………………… 32 TOPIC 7. Computer architecture…………………………………… TOPIC 8. Conditionals………………………………….…………. TOPIC 9. Storage and retrieval of Data…………………….……… Topic 10.Computer Organisation……………………….…………. Vocabulary ………………………………………….……….……. 35 39 41 46 49 LITERATURE USED.……………………………………..……… 75 5 TOPIC 1. PROGRAMMING LANGUAGES Computer languages are generally classed as being “high-level” (like Pascal, Fortran, Ada, Modula-2, Oberon, C or C++) or “lowlevel” (like ASSEMBLER). High-level languages may further be classified as “imperative” (like all of those just mentioned), or “functional” (like Lisp, Scheme, ML, or Haskell), or “logic” (like Prolog). High-level languages are claimed to possess several advantages over low-level ones: • Readability: A good high-level language will allow programs to be written that in some ways resemble a quasi-English description of the underlying algorithms. If care is taken, the coding may be done in a way that is essentially self-documenting, a highly desirable property when one considers that many programs are written once, but possibly studied by humans many times thereafter. • Portability: High-level languages, being essentially machine independent, hold out the promise of being used to develop portable software. This is software that can, in principle (and even occasionally in practice), run unchanged on a variety of different machines — provided only that the source code is recompiled as it moves from machine to machine. To achieve machine independence, high-level languages may deny access to low-level features, and are sometimes spurned by programmers who have to develop low-level machine dependent systems. However, some languages, like C and Modula-2, were specifically designed to allow access to these features from within the context of high-level constructs. • Structure and object orientation: There is general agreement that the structured programming movement of the 1960’s and the object-oriented movement of the 1990’s have resulted in a great improvement in the quality and reliability of code. High-level languages can be designed so as to encourage or even subtly enforce these programming paradigms. • Generality: Most high-level languages allow the writing of a wide variety of programs, thus relieving the programmer of the need to become expert in many diverse languages. 6 • Brevity: Programs expressed in high-level languages are often considerably shorter (in terms of their number of source lines) than their low-level equivalents. • Error checking: Being human, a programmer is likely to make many mistakes in the development of a computer program. Many highlevel languages — or at least their implementations — can, and often do, enforce a great deal of error checking both at compile-time and at run-time. For this they are, of course, often criticized by programmers who have to develop time-critical code, or who want their programs to abort as quickly as possible. These advantages sometimes appear to be overrated, or at any rate, hard to reconcile with reality. For example, readability is usually within the confines of a rather stilted style, and some beginners are disillusioned when they find just how unnatural a high-level language is. Similarly, the generality of many languages is confined to relatively narrow areas, and programmers are often dismayed when they find areas (like string handling in standard Pascal) which seem to be very poorly handled. The explanation is often to be found in the close coupling between the development of high-level languages and of their translators. When one examines successful languages, one finds numerous examples of compromise, dictated largely by the need to accommodate language ideas to rather uncompromising, if not unsuitable, machine architectures. To a lesser extent, compromise is also dictated by the quirks of the interface to established operating systems on machines. Finally, some appealing language features turn out to be either impossibly difficult to implement, or too expensive to justify in terms of the machine resources needed. It may not immediately be apparent that the design of Pascal (and of several of its successors such as Modula2 and Oberon) was governed partly by a desire to make it easy to compile. It is a tribute to its designer that, in spite of the limitations which this desire naturally introduced, Pascal became so popular, the model for so many other languages and extensions, and encouraged the development of superfast compilers such as those found in Borland’s Turbo Pascal and Delphi systems. 7 Exercise 1. Find in text the English for: більша частина; мати перспективу; з точки зору; в усякому разі; якщо потурбуватися; розчаровувати; привабливий; за умови, що; треба віддати належне; узгоджувати(ся) з реальністю. Exercise 2. Learn the words below, then read and translate text 2. judgement — розсудливість, здоровий глузд controversy — суперечка, дискусія, полеміка exacerbate — посилювати, поглиблювати typify — бути типовим представником, уособлювати tempting — спокусливий, привабливий cripple — завдавати шкоди, калічити 154 conceivable — мислимий orthogonality — ортогональність host — велика кількість, безліч blemish — недолік, вада vulnerable — вразливий modularity — модульність selfconsistency — самоузгоджуваність, несуперечність Text 2 SOME ASPECTS OF LANGUAGE DESIGN The design of a programming language requires a high degree of skill and judgement. There is evidence to show that one’s language is not only useful for expressing one’s ideas. Because language is also used to formulate and develop ideas, one’s knowledge of language largely determines how and, indeed, what one can think. In the case of programming languages, there has been much controversy over this. For example, in languages like Fortran — for long the lingua franca of the scientific computing community — recursive algorithms were “difficult” to use (not impossible, just difficult!), with the result that many programmers brought up on Fortran found recursion strange and difficult, even something to be avoided at all costs. It is true that recursive algorithms are sometimes “inefficient”, and that compilers for languages which allow recursion may exacerbate this; on the other hand it is also true that some algorithms are more simply explained in a recursive way than in one which depends on explicit repetition (the best examples probably being those associated with tree manipulation). There are two divergent schools of thought as to how programming languages should be designed. The one, typified by the Wirth school, stresses that languages should be small and understandable, 8 and that much time should be spent in consideration of what tempting features might be omitted without crippling the language as a vehicle for system development. The other, beloved of languages designed by committees with the desire to please everyone, packs a language full of every conceivable potentially useful feature. Both schools claim success. The Wirth school has given us Pascal, Modula-2 and Oberon, all of which have had an enormous effect on the thinking of computer scientists. The other approach has given us Ada, C and C++, which are far more difficult to master well and extremely complicated to implement correctly, but which claim spectacular successes in the marketplace. Other aspects of language design that contribute to success include the following: • Orthogonality: Good languages tend to have a small number of well thought out features that can be combined in a logical way to supply more powerful building blocks. Ideally these features should not interfere with one another, and should not be hedged about by a host of inconsistencies, exceptional cases and arbitrary restrictions. Most languages have blemishes — for example, in Wirth’s original Pascal a function could only return a scalar value, not one of any structured type. Many potentially attractive extensions to well-established languages prove to be extremely vulnerable to unfortunate oversights in this regard. • Familiar notation: Most computers are “binary” in nature. Blessed with ten toes on which to check out their number-crunching programs, humans may be somewhat relieved that high-level languages usually make decimal arithmetic the rule, rather than the exception, and provide for mathematical operations in a notation consistent with standard mathematics. When new languages are proposed, these often take the form of derivatives or dialects of well-established ones, so that programmers can be tempted to migrate to the new language and still feel largely at home — this was the route taken in developing C++ from C, Java from C++, and Oberon from Modula-2, for example. Besides meeting the ones mentioned above, a successful modern high-level language will have been designed to meet the following additional criteria: 9 • Clearly defined: It must be clearly described, for the benefit of both the user and the compiler writer. • Quickly translated: It should admit quick translation, so that program development time when using the language is not excessive. • Modularity: It is desirable that programs can be developed in the language as a collection of separately compiled modules, with appropriate mechanisms for ensuring self-consistency between these modules. • Efficient: It should permit the generation of efficient object code. • Widely available: It should be possible to provide translators for all the major machines and for all the major operating systems. The importance of a clear language description or specification cannot be over-emphasized. This must apply, firstly, to the so-called syntax of the language — that is, it must specify accurately what form a source program may assume. It must apply, secondly, to the so called static semantics of the language — for example, it must be clear what constraints must be placed on the use of entities of differing types, or the scope that various identifiers have across the program text. Finally, the specification must also apply to the dynamic semantics of programs that satisfy the syntactic and static semantic rules — that is, it must be capable of predicting the effect any program expressed in that language will have when it is executed. Exercise 3. Find in text 2 the English for: вразливий; філософський напрям; суперечка; безліч протиріч; мислимий; мати успіх; прикрий недогляд; за будь-яку ціну; щодо цього. Exercise 4. Translate into English paying special attention to the italicized words 1. Цей студент має перспективу стати гарним програмістом за умови, що наполегливо працюватиме. 2. Ми були розчаровані, дізнавшись, що більша частина отриманої інформації не відповідала дійсності (не узгоджувалася з реальністю). 3. Треба віддати належне розробникам цього літака у тому, що він має успіх. 4. Через прикрий недогляд на початковій стадії (at the outset of) розробки цієї системи вона виявилася вразливою до 10 багатьох зовнішніх факторів. 5. Зараз не існує суперечок щодо того, що бортові обчислювальні системи повинні розроблятися передусім в інтересах забезпечення надійності авіаційної техніки та безпеки польоту. 6. У будьякому разі важливість цього проекту не треба переоцінювати, яким би привабливим він не видавався. 7. Поставлене завдання треба виконати за будь-якою ціню. Жоден із нас не має ніяких сумнівів щодо цього. 8. Якщо докласти певних зусиль, то більшу частину недоліків даного проекту можна усунути вже найближчим часом. 9. У межах даного філософського напряму існує безліч протиріч щодо концепцій подальшого розвитку інженерії програмного забезпечення. 10. Запропоноване технічне рішення видається досить привабливим передусім із точки зору можливості протидіяти будьяким мислимим втручанням ззовні.