Low level programming language computer science essay

Technology, Computer

Low-level programming language is a programming language that provides little or no abstraction from a computers instruction set architecture. The word low refers to the small or nonexistent amount of abstraction between the language and machine language; because of this, low-level languages are sometimes described as being “ close to the hardware.”

A low-level language does not need a compiler or interpreter to run. The processor for which the language was written is able to run the code without using either of these.

By comparison, a high-level programming language isolates the execution semantics of computer architecture from the specification of the program, making the process of developing a program simpler and more understandable.

Low-level programming languages are sometimes divided into two categories: first generation, and second generation.

First generation

The first-generation programming language, or 1GL, is a machine code. It is the only language a microprocessor can process directly without a previous transformation. Currently, programmers almost never write programs directly in machine code, because it requires attention to numerous details which a high-level language would handle automatically, and also requires memorizing or looking up numerical codes for every instruction that is used. For this reason, second generation programming languages provide one abstraction level on top of the machine code.

Example: A function in 32-bit x86 machine code to calculate the nth Fibonacci number:

8B542408 83FA0077 06B80000 0000C383

FA027706 B8010000 00C353BB 01000000

B9010000 008D0419 83FA0376 078BD98B

C84AEBF1 5BC3

Second generation

Second-generation programming language, or 2GL, is an assembly language. It is considered a second-generation language because while it is not a microprocessor's native language, an assembly language programmer must still understand the microprocessor's unique registers and instructions. These simple instructions are then assembled directly into machine code. The assembly code can also be abstracted to another layer in a similar manner as machine code is abstracted into assembly code.

Example: The same Fibonacci number calculator as on page one, but in x86 assembly language using MASM syntax:

```
fib:
mov edx, [esp+8]
cmp edx, 0
ja @f
mov eax, 0
ret
@@:
```

```
cmp edx, 2
ja @f
mov eax, 1
ret
@@:
push ebx
mov ebx, 1
mov ecx, 1
@@:
lea eax, [ebx+ecx]
cmp edx, 3
jbe @f
mov ebx, ecx
mov ecx, eax
dec edx
jmp @b
@@:
pop ebx
ret
```

High level programming language

High-level programming language is a programming language with strong abstraction from the details of the computer. In comparison to low-level programming languages, it may use natural language elements, it is easier to use and more portable across platforms. Such languages hide the details

of CPU operations such as memory access models and management of scope.

This greater abstraction and hiding of details is generally intended to make the language user-friendly, as it includes concepts from the problem domain instead of those of the machine used. A high-level language isolates the execution semantics of computer architecture from the specification of the program, making the process of developing a program simpler and more understandable with respect to a low-level language. The amount of abstraction provided defines how “ high-level” a programming language is.

The term “ high-level language” does not imply that the language is superior to low-level programming languages – in fact, in terms of the depth of knowledge of how computers work required to productively program in a given language, the inverse may be true. Rather, “ high-level language” refers to the higher level of abstraction from machine language. Rather than dealing with registers, memory addresses and call stacks, high-level languages deal with usability, threads, locks, objects, variables, arrays and complex arithmetic or Boolean expressions. In addition, they have no opcodes that can directly compile the language into machine code, unlike low-level assembly language. Other features such as string handling routines, object-oriented language features and file input/output may also be present.

High-level languages make complex programming simpler, while low-level languages tend to produce more efficient code. High-level programming

features like more generic data structures, run-time interpretation and intermediate code files often result in slower execution speed, higher memory consumption and larger binary size. For this reason, code which needs to run particularly quickly and efficiently may be written in a lower-level language, even if a higher-level language would make the coding easier.

With the growing complexity of modern microprocessor architectures, well-designed compilers for high-level languages frequently produce code comparable in efficiency to what most low-level programmers can produce by hand, and the higher abstraction may allow for more powerful techniques providing better overall results than their low-level counterparts in particular settings.

There are three models of execution for modern high-level languages:

Interpreted – Interpreted languages are read and then executed directly, with no compilation stage.

CompiledA – Compiled languages are transformed into an executable form before running. There are two types of compilation: Intermediate representationsA – When a language is compiled to an intermediate representation, that representation can be optimized or saved for later execution without the need to re-read the source file. When the intermediate representation is saved it is often represented as bytecode. Machine code generation – Some compilers compile source code directly into machine code. Virtual machines that execute bytecode directly or transform it further

into machine code have blurred the once clear distinction between intermediate representations and truly compiled languages.

Translated – A language may be translated into a low-level programming language for which native code compilers are already widely available. The C programming language is a common target for such translators.

Examples of high level programming language include:

Java, C, Python, Scheme, Prolog, C++, C#, VB, Java Script, Ruby and Lisp.

Comparison of high and low programming languages – below are similar programs in both languages to find greatest and smallest data value in a data set.

Validating raw data

Input validation is an important part of any computer application that requires user interaction. It applies to anything that the application does to ensure that data entered by the user is acceptable for the purposes of the application.

Input validation can take place at various times in the data entry cycle. For example, the programmer can:

Constrain the user’s entry of data before it begins by providing very restricted data input fields that permit only valid choices. The most common way to do this is to provide standard controls that do not permit free keyboard entry, such as drop-down lists, option buttons, and check boxes.

Constrain the user’s entry of data at the moment that it occurs by monitoring every keystroke for validity and rejecting unwanted input as it’s typed. For instance, a particular entry field might seem to the user to ignore anything but numeric characters.

React to the user’s entry of data after the user is finished, accepting or rejecting the contents of a data field when the user attempts to leave the field or close the screen.

Input validation can also have varying degrees of user participation. For instance, the program can

Automatically correct a user’s mistakes without asking the user’s opinion.

Warn the user of incorrect input and prompt the user to correct the input before allowing the user to continue with other activities.

Benifets of Data Validation: Reduces the time that is spent completing forms and eliminates costs associated with errors by validating data, improving efficiency and minimizing the high cost of exception handling resulting from data input errors. Validation happens immediately in a visual basic form and can catch common errors such as not entering a required field, incorrect data type or entering incorrect data based on other data previously entered into the form.

Example – validation of checking if the date entered is after today’s date:

Private Sub Date_Entered_AfterUpdate()

A A A If Me. Date_Entered> date() thenA A A A A A A ‘ Date_Entered is form field name

A A A A A A A A A A A msgbox “ Please enter a date less than or equal to today’s date.”

A A A A A A A A A A A Me. Date_Entered. setfocusA A A A A ‘ set cursor back in the date field

A A A end if

End Sub

Using strings with passwords in visual basic

The use of strings benefits data entry in a password application in visual basic because the program would not reset itself and crash on error input. This could affect the safety of the password which could be compromised.

It compares the 4 digit numbers inputted in order and will only allow access when all 4 characters are correct and in the right sequence. If this doesn’t happen it will reset and wipe the numbers clear. This is so the imputer cannot see which one’s he/she has got right and try different ones for the others remaining.

In visual basic for the above program you could limit the user to a certain amount of attempts before the user is locked out. When the code is incorrect you could have a warning flash up on the computer, or if the wrong sort of data is inputted you can have a warning asking for the correct data. Likewise

if you have not inputted 4 digits it would crash so this can be modified in the code strings with ” “.

Data Types

Boolean Data TypeA

This data type holds values that can be only true or false. The keywords True and False correspond to the two states of Boolean variables. Use the Boolean data type to contain two-state values such as true/false, yes/no, or on/off. The default value of Boolean is False.

Type Conversions

When Visual Basic converts numeric data type values to Boolean, 0 becomes False and all other values become true. When Visual Basic converts Boolean values to numeric types, False becomes 0 and True becomes -1.

When you convert between Boolean values and numeric data types, the . NET Framework conversion methods do not always produce the same results as the Visual Basic conversion keywords. This is because the Visual Basic conversion retains behaviour compatible with previous versions.

Programming guide

Negative Numbers. Boolean is not a numeric type and cannot represent a negative value. In any case, you should not use Boolean to hold numeric values.

Type Characters. Boolean has no literal type character or identifier type character.

Framework Type. The corresponding type in the . NET Framework is the system Boolean structure.

In the following example, runningVB is a Boolean variable, which stores a simple yes/no setting.

Dim runningVB As Boolean

‘ Check to see if program is running on Visual Basic engine.

If scriptEngine = “ VB” Then

runningVB = True

End If

Integer Data TypeA

Integer data holds signed 32-bit (4-byte) integers ranging in value from -2, 147, 483, 648 through 2, 147, 483, 647.

The Integer data type provides optimal performance on a 32-bit processor. The other integral types are slower to load and store from and to memory. The default value of Integer is 0.

Programming guide

Interop Considerations. If you are interfacing with components not written for the . NET Framework, for example Automation or COM objects, keep in mind that Integer has a different data width (16 bits) in other environments.

Widening. The Integer data type widens to Long, Decimal, Single, or Double. This means you can convert Integer to any of these types without encountering a System. OverflowException error.

Type Characters. Appending the literal type character I to a literal forces it to the Integer data type. Appending the identifier type character % to any identifier forces it to Integer.

Framework Type. The corresponding type in the . NET Framework is the System. Int32 structure.

If you try to set a variable of an integral type to a number outside the range for that type, an error occurs. If you try to set it to a fraction, the number is rounded. The following example shows this.

‘ The valid range of an Integer variable is -2147483648 through +2147483647.

Dim k As Integer

‘ The following statement causes an error because the value is too large.

k = 2147483648

‘ The following statement sets k to 6.

k = CInt(5. 9)

Char Data TypeA

Holds unsigned 16-bit (2-byte) code points ranging in value from 0 through 65535. Each code point, or character code, represents a single Unicode character.

You use the Char data type when you need to hold only a single character and do not need the overhead of String. In some cases you can use Char(), an array of Char elements, to hold multiple characters. The default value of Char is the character with a code point of 0.

Unicode Characters

The first 128 code points (0-127) of Unicode correspond to the letters and symbols on a standard U. S. keyboard. These first 128 code points are the same as those the ASCII character set defines. The second 128 code points (128-255) represent special characters, such as Latin-based alphabet letters, accents, currency symbols, and fractions. Unicode uses the remaining code points (256-65535) for a wide variety of symbols, including worldwide textual characters, diacritics, and mathematical and technical symbols.

You can use methods like IsDigit and IsPunctuation on a Char variable to determine its Unicode classification.

Type Conversions

Visual Basic does not convert directly between Char and the numeric types. You can use the Asc, AscW Functions to convert a Char value to an Integer representing its code point. You can use the Chr, ChrW Functions to convert an Integer value to a Char having that code point.

If the type checking switch (Option Strict Statement) is on, you must append the literal type character to a single-character string literal to identify it as the Char data type. The following example on page 8 illustrates this.

Option Strict On

Dim charVar As Char

‘ The following statement attempts to convert a String literal to Char.

‘ Because Option Strict is On, it generates a compiler error.

charVar = “ Z”

‘ The following statement succeeds because it specifies a Char literal.

charVar = “ Z” C

Programming guide

Negative Numbers. Char is an unsigned type and cannot represent a negative value. In any case, you should not use Char to hold numeric values.

Interop Considerations. If you are interfacing with components not written for the . NET Framework, for example Automation or COM objects.

Widening. The Char data type widens to String. This means you can convert Char to String without encountering a System. OverflowException error.

Type Characters. Appending the literal type character C to a single-character string literal forces it to the Char data type. Char has no identifier type character.

Framework Type. The corresponding type in the . NET Framework is the System. Char structure

Comparison table between Ladder Logic and Visual Basic

Language

Ladder Logic

Visual Basic

Suitability for engineering applications

High

Medium

Availability

Medium

High

User friendliness

Low

High

Cost of software

High

Low

Size of code

Low (high compact)

High

Difficulty of use

Medium to High

Easy

Ease of programming

Medium

User has knowledge then low User has no knowledge then High

Ideal use

Machine control

Simulation

Resources:

www. wikipedia. org, www. msdn. microsoft. com, www. fortran. com, visual basic help, www. visualbasic. freetubes. net, www. blueclaw-db. com