DATA TYPES, VARIABLES, AND ARITHMETIC pdf

1: Definitions, Uses, Data Types, and Levels of Measurement

In this chapter from Programming in C, 4th Edition, Stephen G. Kochan covers the int, float, double, char, and _Bool data types, modifying data types with short, long, and long long, the rules for naming variables, basic math operators and arithmetic expressions, and type casting.

The result of is handed to the printf routine. This helps make the program more readable. You also might have noticed in each program presented thus far that a blank space was placed around each operator. This, too, is not required and is done solely for aesthetic reasons. In general, you can add extra blank spaces just about anywhere that a single blank space is allowed. A few extra presses of the spacebar prove worthwhile if the resulting program is easier to read. The expression in the first printf call of Program 3. Evaluation of this expression proceeds as follows: Because division has higher precedence than addition, the value of a 25 is divided by 5 first. This gives the intermediate result of 5. Because multiplication also has higher precedence than addition, the intermediate result of 5 is next multiplied by 2, the value of b, giving a new intermediate result of Finally, the addition of 6 and 10 is performed, giving a final result of The second printf statement introduces a new twist. You would expect that dividing a by b and then multiplying by b would return the value of a, which has been set to But this does not seem to be the case, as shown by the output display of It might seem like the computer lost a bit somewhere along the way. The fact of the matter is that this expression was evaluated using integer arithmetic. If you glance back at the declarations for the variables a and b, you will recall that they were both declared to be of type int. Whenever a term to be evaluated in an expression consists of two integers, the C system performs the operation using integer arithmetic. In such a case, all decimal portions of numbers are lost. Therefore, when the value of a is divided by the value of b, or 25 is divided by 2, you get an intermediate result of 12 and not In addition, keep in mind that no rounding occurs, the decimal value is simply dropped, so integer division that ends up with As you can see from the next-to-last printf statement in Program 3. The resulting program is more efficientâ€”that is, it executes more quickly on many computers. On the other hand, if you need the decimal place accuracy, the choice is clear. The only question you then must answer is whether to use a float, double, or long double. The answer to this question depends on the desired accuracy of the numbers you are dealing with, as well as their magnitude. In the last printf statement, the value of the variable a is negated by use of the unary minus operator. A unary operator is one that operates on a single value, as opposed to a binary operator, which operates on two values. The minus sign actually has a dual role: As a binary operator, it is used for subtracting two values; as a unary operator, it is used to negate a value. Once again, in Appendix A you will find a table summarizing the various operators and their precedences. Try to determine how this operator works by analyzing Program 3. For a reminder, before a series of statements that use the modulus operator are printed, the first printf statement prints the values of the four variables used in the program. In the first example, the remainder after 25 is divided by 5 and is displayed as 0. If you divide 25 by 10, you get a remainder of 5, as verified by the second line of output. Dividing 25 by 7 gives a remainder of 4, as shown in the third output line. The last line of output in Program 3. First, you will notice that the program statement has been written on two lines. This is perfectly valid in C. In fact, a program statement can be continued to the next line at any point at which a blank space could be used. The continuation of the printf call in Program 3. Turn your attention to the expression evaluated in the final statement. You will recall that any operations between two integer values in C are performed with integer arithmetic. Therefore, any remainder resulting from the division of two integer values is simply discarded. Multiplying this value by the value of d, which is 7, produces the intermediate result of It is no coincidence that this value is the same as the value of the variable a. As far as precedence is concerned, the modulus operator has equal precedence to the multiplication and division operators. You should note that some compilers might give warning messages to alert you of the fact that conversions are being performed. So, when the value of f1 is assigned to i1 in the previous program, the number Assigning an integer variable to a floating variable does not cause any change in the value of the number; the value is simply converted by the system and stored in the floating variable. The first has to do with integer arithmetic,

which was previously discussed in this chapter. Whenever two operands in an expression are integers and this applies to short, unsigned, long, and long long integers as well , the operation is carried out under the rules of integer arithmetic. Therefore, any decimal portion resulting from a division operation is discarded, even if the result is assigned to a floating variable as you did in the program. Therefore, when the integer variable i2 is divided by the integer constant , the system performs the division as an integer division. The next division performed in the previous listing involves an integer variable and a floating-point constant. Any operation between two values in C is performed as a floating-point operation if either value is a floating-point variable or constant. Therefore, when the value of i2 is divided by The type cast operator has the effect of converting the value of the variable i2 to type float for purposes of evaluation of the expression. In no way does this operator permanently affect the value of the variable i2; it is a unary operator that behaves like other unary operators. The type cast operator has a higher precedence than all the arithmetic operators except the unary minus and unary plus. Of course, if necessary, you can always use parentheses in an expression to force the terms to be evaluated in any desired order. As another example of the use of the type cast operator, the expression int The Assignment Operators The C language permits you to join the arithmetic operators with the assignment operator using the following general format: In addition, op can be any of the bit operators for shifting and masking, which is discussed later. So, the previous statement is equivalent to this statement: The addition is performed first because the addition operator has higher precedence than the assignment operator. In fact, all operators but the comma operator have higher precedence than the assignment operators, which all have the same precedence. In this case, this expression is identical to the following: First, the program statement becomes easier to write because what appears on the left side of the operator does not have to be repeated on the right side. Second, the resulting expression is usually easier to read. Third, the use of these operators can result in programs that execute more quickly because the compiler can sometimes generate less code to evaluate an expression. The best way to know if your compiler supports these types is to examine the summary of data types in Appendix A.

DATA TYPES, VARIABLES, AND ARITHMETIC pdf

2: Variables and Data Types

Data types, Variables and Arithmetic Operators. Let us write a simple equation in math to calculate mean of a set of numbers: a = 15, b = 35, c =

In general we write: Your teacher has a particular fondness for this symbol since the first computer he had much access to had that nickname. There are three important rules for using the summation operator: Since multiplication distributes over addition, the sum of a constant times a set of numbers is the same as the constant times the sum of the set of numbers. Joe got scores of and for his verbal and quantitative SAT scores whereas Jim got scores of and , respectively. In addition to the operations of addition, subtraction, multiplication, and division, several other arithmetic operators often appear. Exponentiation and absolute value are two such. Also, various symbols of inclusion parentheses, brackets, braces, vincula are used. When the square root symbol surd and symbol of inclusion, in recent history a vinculum, but historically parentheses is used, we general although not quite always mean only the positive square root. The absolute value operator indicates the distance always non-negative a number is from the origin zero. The symbol used is a vertical line on either side of the operand. There is a proscribed order for arithmetic operations to be performed. Some calculators are algebraic and handle this appropriately, others do not. Parentheses and other symbols of inclusion are used to modify the normal order of operations. We say these symbols of inclusion have the highest priority or precidence. Exponentiation is done next. There is confusion when exponents are stacked which we will not deal with here except to say computer scientists tend to do it from left to right while mathematicians know that is wrong. Multiplication and Division are done next, in order, from left to right. Addition and Subtraction are done next, in order, from left to right. Precision The distinction between accuracy and precision, reviewed in Numbers lesson 9 , is very important. This ties in with significant figures, and proper rounding of results. I have several major concerns regarding significant digits. There needs to be sufficient not to few. Slide rule accuracy or three significant digits has a long-standing precident in science. We are not doing science here so two may suffice, but rarely one. There should not be too many significant digits. Generally, more than 5 is probably a joke, especially in the "softer" sciences. Care must be taken so that a primary statistics such as variance is not incorrectly derived from a secondary statistic such as standard deviation in such a way that accuracy is lost. We will discuss this more in textbook Chapter 3. A mean and standard deviation or mean and margin of error should be given to compatible precision. There are proper rules, but they are difficult to explain to the general public. Thus every statistics book gives its own heuristic. Uses and Abuses of Statistics Most of the time, samples are used to infer something draw conclusions about the population. If an experiment or study was done cautiously and results were interpreted without bias , then the conclusions would be accurate. However, occasionally the conclusions are inaccurate or inaccurately portrayed for the following reasons: Sample is too small. Even a large sample may not represent the population. Unauthorized personnel are giving wrong information that the public will take as truth. A possibility is a company sponsoring a statistics research to prove that their company is better. Visual aids may be correct, but emphasize different aspects. Often a chart will use a symbol which is both twice as long and twice as high to represent something twice as much. The area, in this case however, is four times as much! Precise statisitics or parameters may incorrectly convey a sense of high accuracy. Misleading or unclear percentages are often used. Statistics are often abused. Many examples could be added, even books have been written but it will be more instructive and fun to find them on your own. Types of Data A dictionary defines data as facts or figures from which conclusions may be drawn. Thus, technically, it is a collective, or plural noun. Some recent dictionaries acknowledge popular usage of the word data with a singular verb. My mother and step-mother were both English teachers, so clearly no offense is intended above. Datum is the singular form of the noun data. Data can be classified as either numeric or nonnumeric. Specific terms are used as follows: Qualitative data are nonnumeric. Qualitative data are often termed catagorical data. Some books use the terms individual and variable to reference the objects and characteristics described by a set of data. They also stress the importance of exact definitions of these variables, including what units they are recorded in.

DATA TYPES, VARIABLES, AND ARITHMETIC pdf

The reason the data were collected is also important. Quantitative data are numeric. Quantitative data are further classified as either discrete or continuous. Discrete data are numeric data that have a finite number of possible values. Another classic is the spin or electric charge of a single electron. Quantum Mechanics, the field of physics which deals with the very small, is much concerned with discrete values. When data represent counts, they are discrete. An example might be how many students were absent on a given day. Counts are usually considered exact and integer. Continuous data have infinite possibilities: The real numbers are continuous with no gaps or interruptions. Physically measureable quantities of length, volume, time, mass, etc. At the physical level microscopically , especially for mass, this may not be true, but for normal life situations is a valid assumption. The structure and nature of data will greatly affect our choice of analysis method. By structure we are referring to the fact that, for example, the data might be pairs of measurements. Consider the legend of Galileo dropping weights from the leaning tower of Pisa. The times for each item would be paired with the mass and surface area of the item. Something which Galileo clearly did was measure the time it took a pendulum to swing with various amplitudes. Galileo Galilei is considered a founder of the experimental method. Levels of Measurement The experimental scientific method depends on physically measuring things. The concept of measurement has been developed in conjunction with the concepts of numbers and units of measurement. Statisticians categorize measurements according to levels. Each level corresponds to how this measurement can be treated mathematically. Nominal data have no order and thus only gives names or labels to various categories. Ordinal data have order, but the interval between measurements is not meaningful. Interval data have meaningful intervals between measurements, but there is no true starting point zero. Ratio data have the highest level of measurement. Ratios between measurements as well as intervals are meaningful because there is a starting point zero. Nominal comes from the Latin root nomen meaning name. Nomenclature, nominative, and nominee are related words. Gender is something you are born with, whereas sex is something you should get a license for. Colors To most people, the colors: To an electronics student familiar with color-coded resistors, this data is in ascending order and thus represents at least ordinal data. To a physicist, the colors: Temperatures What level of measurement a temperature is depends on which temperature scale is used. Only Kelvin and Rankine have true zeroes starting point and ratios can be found. Celsius and Fahrenheit are interval data; certainly order is important and intervals are meaningful. Rankine has the same size degree as Fahrenheit but is rarely used. To interconvert Fahrenheit and Celsius, see Numbers lesson Note that since , the use of the degree symbol on tempertures Kelvin is no longer proper. Although ordinal data should not be used for calculations, it is not uncommon to find averages formed from data collected which represented Strongly Disagree, Also, averages of nominal data zip codes, social security numbers is rather meaningless!

DATA TYPES, VARIABLES, AND ARITHMETIC pdf

3: Types, Operators, and Expressions - Dynamic Tracing Guide

3. Variables, Data Types, and Arithmetic Expressions. The true power of programs you create is their manipulation of data. In order to truly take advantage of this power, you need to better understand the different data types you can use, as well as how to create and name variables.

Bitwise arithmetic operators You may be aware already that computers store all data internally in binary. This is also the case with all the data types in MSX-C. As an example, the table below shows how some integer values are represented internally in the MSX computer memory: However, there are situations when working with binary data makes sense, as in the case of working with graphics or programming hardware devices. C provides several bitwise operators that we can use to operate with values bit by bit: We can summarize the way these work with these very simple tables: This can be used to turn off especific bits in a value. In an OR operation, the result is 1 if either operand, or both, is 1: We can use this property to turn on bits in a value: XOR can be used to invert especific bits of a value: It is also used extensively in cryptographic applications. This operator inverts all the bits in the operand at the same time: Arithmetic shift to the left: For example, the code: The lower bits are filled with zeros. This operator is very useful to perform fast multiplication: Arithmetic shift to the right: Again, the bits that fall outside to the right are discarded, and zeroes are introduced to the left. Shifting one bit to the right is the same as dividing by two; two bits to the right is the same as dividing by four, and so on. The answer is simple: The reason for this is that internally, everything in C is a number. This is done via library functions that we will see in another post. In the next post we will see how to actually read single characters and text strings from the keyboard. This series of articles is supported by your donations. Every little amount helps.

DATA TYPES, VARIABLES, AND ARITHMETIC pdf

4: C data types - Wikipedia

(T/F) A cast operator, when applied to an arithmetic expression in parentheses, converts every constant and variable in the expression into the specified type. T (T/F) A numeric literal constant is always treated as a value of the double type.

The binary operator is used to set bits in an integer operand. The shift operators are used to move bits left or right in a given integer operand. Shifting left fills empty bit positions on the right-hand side of the result with zeroes. Shifting right using an unsigned integer operand fills empty bit positions on the left-hand side of the result with zeroes. Shifting right using a signed integer operand fills empty bit positions on the left-hand side with the value of the sign bit, also known as an arithmetic shift operation. Shifting an integer value by a negative number of bits or by a number of bits larger than the number of bits in the left-hand operand itself produces an undefined result. The D compiler will produce an error message if the compiler can detect this condition when you compile your D program. Assignment Operators D provides the following binary assignment operators for modifying D variables. You can only modify D variables and arrays. Kernel data objects and constants may not be modified using the D assignment operators. These assignment operators obey the same rules for operand types as the binary forms described earlier. The result of any assignment operator is an expression equal to the new value of the left-hand expression. You can use the assignment operators or any of the operators described so far in combination to form expressions of arbitrary complexity. You can use parentheses to group terms in complex expressions. These operators can only be applied to variables, and may be applied either before or after the variable name. If the operator appears before the variable name, the variable is first modified and then the resulting expression is equal to the new value of the variable. For example, the following two expressions produce identical results: The increment and decrement operators can be applied to integer or pointer variables. When applied to integer variables, the operators increment or decrement the corresponding value by one. When applied to pointer variables, the operators increment or decrement the pointer address by the size of the data type referenced by the pointer. Pointers and pointer arithmetic in D are discussed in Pointers and Arrays. Conditional Expressions Although D does not provide support for if-then-else constructs, it does provide support for simple conditional expressions using the? These operators enable a triplet of expressions to be associated where the first expression is used to conditionally evaluate one of the other two. For example, the following D statement could be used to set a variable x to one of two strings depending on the value of i: If the first expression is true, the second expression is evaluated and the?: If the first expression is false, the third expression is evaluated and the?: As with any D operator, you can use multiple?: For example, the following expression would take a char variable c containing one of the characters , a-z, or A-Z and return the value of this character when interpreted as a digit in a hexadecimal base 16 integer: The second and third expressions may be of any compatible types. You may not construct a conditional expression where, for example, one path returns a string and another path returns an integer. The second and third expressions also may not invoke a tracing function such as trace or printf. If you want to conditionally trace data, use a predicate instead, as discussed in Introduction. Type Conversions When expressions are constructed using operands of different but compatible types, type conversions are performed in order to determine the type of the resulting expression. These rules are sometimes referred to as the usual arithmetic conversions. A simple way to describe the conversion rules is as follows: If a conversion is required, the operand of lower rank is first promoted to the type of higher rank. Promotion does not actually change the value of the operand: If an unsigned operand is promoted, the unused high-order bits of the resulting integer are filled with zeroes. If a signed operand is promoted, the unused high-order bits are filled by performing sign extension. If a signed type is converted to an unsigned type, the signed type is first sign-extended and then assigned the new unsigned type determined by the conversion. Integers and other types can also be explicitly cast from one type to another. In D, pointers and integers can be cast to any integer or pointer types, but not to other types. Rules for casting and promoting strings and character arrays are discussed in Strings. An integer or pointer cast is formed using an expression such as: Integers are cast to types of higher rank by performing promotion. Integers are cast to types of lower rank by

zeroing the excess high-order bits of the integer. Because D does not permit floating-point arithmetic, no floating-point operand conversion or casting is permitted and no rules for implicit floating-point conversion are defined. Precedence The D rules for operator precedence and associativity are described in the following table. These rules are somewhat complex, but are necessary to provide precise compatibility with the ANSI-C operator precedence rules. The table entries are in order from highest precedence to lowest precedence. D Operator Precedence and Associativity Operators.

DATA TYPES, VARIABLES, AND ARITHMETIC pdf

5: c-language data type arithmetic rules - Stack Overflow

the variable belongs to the class and the variable exists from the start of the program modulus or a remainder the type of the results is determined by the ____ of the operands not their ____this rule applies to all intermidiate results in expressions.

In order to truly take advantage of this power, you need to better understand the different data types you can use, as well as how to create and name variables. C has a rich variety of math operators that you can use to manipulate your data. In this chapter you will cover: As you will recall, a variable declared to be of type int can be used to contain integral values onlyâ€”that is, values that do not contain decimal places. The C programming language provides four other basic data types: A variable declared to be of type float can be used for storing floating-point numbers values containing decimal places. The double type is the same as type float, only with roughly twice the precision. These one-or-the-other choices are also known as binary choices. In C, any number, single character, or character string is known as a constant. For example, the number 58 represents a constant integer value. The character string "Programming in C is fun. Expressions consisting entirely of constant values are called constant expressions. The Integer Type int In C, an integer constant consists of a sequence of one or more digits. A minus sign preceding the sequence indicates that the value is negative. No embedded spaces are permitted between the digits, and values larger than cannot be expressed using commas. So, the value 12, is not a valid integer constant and must be written as Two special formats in C enable integer constants to be expressed in a base other than decimal base If the first digit of the integer value is a 0, the integer is taken as expressed in octal notationâ€”that is, in base 8. In that case, the remaining digits of the value must be valid base-8 digits and, therefore, must be 0â€”7. So, to express the value 50 in base 8 in C, which is equivalent to the value 40 in decimal, the notation is used. In such a case, the value is displayed in octal without a leading zero. If an integer constant is preceded by a zero and the letter x either lowercase or uppercase , the value is taken as being expressed in hexadecimal base 16 notation. Immediately following the letter x are the digits of the hexadecimal value, which can be composed of the digits 0â€”9 and the letters aâ€”f or Aâ€”F. The letters represent the values 10â€”15, respectively. This range has to do with the amount of storage that is allocated to store a particular type of data. In general, that amount is not defined in the language. For example, an integer might take up 32 bits on your computer, or perhaps it might be stored in You should never write programs that make any assumptions about the size of your data types. You are, however, guaranteed that a minimum amount of storage will be set aside for each basic data type. The Floating Number Type float A variable declared to be of type float can be used for storing values containing decimal places. A floating-point constant is distinguished by the presence of a decimal point. Floating-point constants can also be expressed in scientific notation. The value before the letter e is known as the mantissa, whereas the value that follows is called the exponent. This exponent, which can be preceded by an optional plus or minus sign, represents the power of 10 by which the mantissa is to be multiplied. So, in the constant 2. This constant represents the value 2. Incidentally, the letter e, which separates the mantissa from the exponent, can be written in either lowercase or uppercase. This decision is based on the value of the exponent: A hexadecimal floating constant consists of a leading 0x or 0X, followed by one or more decimal or hexadecimal digits, followed by a p or P, followed by an optionally signed binary exponent. The Extended Precision Type double The double type is very similar to the float type, but it is used whenever the range provided by a float variable is not sufficient. Variables declared to be of type double can store roughly twice as many significant digits as can a variable of type float. Most computers represent double values using 64 bits. Unless told otherwise, all floating-point constants are taken as double values by the C compiler. To explicitly express a float constant, append either an f or F to the end of the number, as follows: The Single Character Type char A char variable can be used to store a single character. The first constant represents the letter a, the second is a semicolon, and the third is the character zeroâ€”which is not the same as the number zero. Do not confuse a character constant, which is a single character enclosed in single quotes, with a character string, which is any number of characters enclosed in double quotes. This is because the backslash character is a special character in the C

system and does not actually count as a character. There are other special characters that are initiated with the backslash character. The precise amount of memory that is used is unspecified. For example, a variable of this type might be used to indicate whether all data has been read from a file. By convention, 0 is used to indicate a false value, and 1 indicates a true value. An example of this is shown in Program 5. In fact, the actual value displayed is dependent on the particular computer system you are using. The reason for this inaccuracy is the particular way in which numbers are internally represented inside the computer. You have probably come across the same type of inaccuracy when dealing with numbers on your pocket calculator. If you divide 1 by 3 on your calculator, you get the result. Theoretically, there should be an infinite number of 3s. But the calculator can hold only so many digits, thus the inherent inaccuracy of the machine. The same type of inaccuracy applies here: When displaying the values of float or double variables, you have the choice of three different formats. Unless told otherwise, printf always displays a float or double value to six decimal places rounded. You see later in this chapter how to select the number of decimal places that are displayed. Once again, six decimal places are automatically displayed by the system. Remember that whereas a character string such as the first argument to printf is enclosed within a pair of double quotes, a character constant must always be enclosed within a pair of single quotes. An example of a long int declaration might be long int factorial; This declares the variable factorial to be a long integer variable. As with floats and doubles, the particular accuracy of a long variable depends on your particular computer system. A constant value of type long int is formed by optionally appending the letter L upper- or lowercase onto the end of an integer constant. No spaces are permitted between the number and the L. To display the value of a long int using printf , the letter l is used as a modifier before the integer format characters i, o, and x. There is also a long long integer data type, so long long int maxAllowedStorage; declares the indicated variable to be of the specified extended accuracy, which is guaranteed to be at least 64 bits wide. The long specifier is also allowed in front of a double declaration, as follows: The specifier short, when placed in front of the int declaration, tells the C compiler that the particular variable being declared is used to store fairly small integer values. The motivation for using short variables is primarily one of conserving memory space, which can be an issue in situations in which the program needs a lot of memory and the amount of available memory is limited. On some machines, a short int takes up half the amount of storage as a regular int variable does. In any case, you are guaranteed that the amount of space allocated for a short int will not be less than 16 bits. There is no way to explicitly write a constant of type short int in C. To display a short int variable, place the letter h in front of any of the normal integer conversion characters: Alternatively, you can also use any of the integer conversion characters to display short ints, due to the way they can be converted into integers when they are passed as arguments to the printf routine. The final specifier that can be placed in front of an int variable is used when an integer variable will be used to store only positive numbers. The declaration unsigned int counter; declares to the compiler that the variable counter is used to contain only positive values. By restricting the use of an integer variable to the exclusive storage of positive integers, the accuracy of the integer variable is extended. An unsigned int constant is formed by placing the letter u or U after the constant, as follows: When declaring variables to be of type long long int, long int, short int, or unsigned int, you can omit the keyword int. Therefore, the unsigned variable counter could have been equivalently declared as follows: The signed qualifier can be used to explicitly tell the compiler that a particular variable is a signed quantity. In later sections of this book, many of these different types are illustrated with actual program examples. Chapter 13 goes into more detail about data types and conversions.

DATA TYPES, VARIABLES, AND ARITHMETIC pdf

6: PHP Lesson 2: PHP Data Types and Variables - www.enganchecubano.com

PHP Data Types and Variables. In this lesson will be talking about PHP Data Types and Variables which is an important topic.. Data Type. Data types identifies the type of the data we are dealing with in our PHP programs (Or in any other programming language.

The standard only requires size relations between the data types and minimum sizes for each data type: The relation requirements are that the long long is not smaller than long, which is not smaller than int, which is not smaller than short. The minimum size for char is 8 bits, the minimum size for short and int is 16 bits, for long it is 32 bits and long long must contain at least 64 bits. The type int should be the integer type that the target processor is most efficiently working with. This allows great flexibility: However, several different integer width schemes data models are popular. Because the data model defines how different programs communicate, a uniform data model is used within a given operating system application interface. POSIX requires char to be exactly eight bits in size. Various rules in the C standard make unsigned char the basic type used for arrays suitable to store arbitrary non-bit-field objects: The only guarantee is that long double is not smaller than double, which is not smaller than float. Usually, the bit and bit IEEE binary floating-point formats are used, if supported by hardware. These types may be wider than long double. C99 also added complex types: This behavior exists to avoid integer overflows in implicit narrowing conversions. For example, in the following code: This is because does not fit in the data type, which results in the lower 8 bits of it being used, resulting in a zero value. However, changing the type causes the previous code to behave normally: It is only guaranteed to be valid against pointers of the same type; subtraction of pointers consisting of different types is implementation-defined. Interface to the properties of the basic types[edit] Information about the actual properties, such as size, of the basic arithmetic types, is provided via macro constants in two headers: The actual values depend on the implementation. Fixed-width integer types[edit] The C99 standard includes definitions of several new integer types to enhance the portability of programs. The new types are especially useful in embedded environments where hardware usually supports only several types and that support varies between different environments. The types can be grouped into the following categories: Exact-width integer types which are guaranteed to have the same number N of bits across all implementations. Included only if it is available in the implementation. Least-width integer types which are guaranteed to be the smallest type available in the implementation, that has at least specified number N of bits. Fastest integer types which are guaranteed to be the fastest integer type available in the implementation, that has at least specified number N of bits. Pointer integer types which are guaranteed to be able to hold a pointer. Maximum-width integer types which are guaranteed to be the largest integer type in the implementation. The following table summarizes the types and the interface to acquire the implementation details N refers to the number of bits:

DATA TYPES, VARIABLES, AND ARITHMETIC pdf

7: Variables, Data Types, and Arithmetic Expressions

Container for data: has a name, type and current www.enganchecubano.comationStatement that defines a variable (name, type, initial variable) and allocates the variable during www.enganchecubano.comteDuring execution, the reservation of memory location(s) represented by a www.enganchecubano.commentStatement that changes the value (current contents) of a variable.

If you cannot initialize your local variable where it is declared, make sure to assign it a value before you attempt to use it. Accessing an uninitialized local variable will result in a compile-time error. Primitive types are special data types built into the language; they are not objects created from a class. A literal is the source code representation of a fixed value; literals are represented directly in your code without requiring computation. It is recommended that you use the upper case letter L because the lower case letter l is hard to distinguish from the digit 1. Values of the integral types byte, short, int, and long can be created from int literals. Values of type long that exceed the range of int can be created from long literals. Integer literals can be expressed by these number systems: Base 10, whose digits consists of the numbers 0 through 9; this is the number system you use every day Hexadecimal: Base 16, whose digits consist of the numbers 0 through 9 and the letters A through F Binary: However, if you need to use another number system, the following example shows the correct syntax. The prefix 0x indicates hexadecimal and 0b indicates binary: The floating point types float and double can also be expressed using E or e for scientific notation , F or f bit float literal and D or d bit double literal; this is the default and by convention is omitted. If your editor and file system allow it, you can use such characters directly in your code. Unicode escape sequences may be used elsewhere in a program such as in field names, for example , not just in char or String literals. The Java programming language also supports a few special escape sequences for char and String literals: Therefore, null is often used in programs as a marker to indicate that some object is unavailable. This refers to the object of type Class that represents the type itself. This feature enables you, for example. For instance, if your code contains numbers with many digits, you can use an underscore character to separate digits in groups of three, similar to how you would use a punctuation mark like a comma, or a space, as a separator. The following example shows other ways you can use the underscore in numeric literals: At the beginning or end of a number Adjacent to a decimal point in a floating point literal Prior to an F or L suffix In positions where a string of digits is expected The following examples demonstrate valid and invalid underscore placements which are highlighted in numeric literals:

DATA TYPES, VARIABLES, AND ARITHMETIC pdf

8: X++ variables and data types - Finance & Operations | Dynamics | #MSDynFO | Microsoft Docs

The compiler looks at the types of the operands for a single operation, and promotes both to the "larger" type (e.g., if one is int and the other double, it'll convert the int to double, then do the operation).

The result of is handed to the printf routine. This helps make the program more readable. You also might have noticed in each program presented thus far that a blank space was placed around each operator. This, too, is not required and is done solely for aesthetic reasons. In general, you can add extra blank spaces just about anywhere that a single blank space is allowed. A few extra presses of the spacebar prove worthwhile if the resulting program is easier to read. The expression in the first printf call of Program 3. Evaluation of this expression proceeds as follows: Because division has higher precedence than addition, the value of a 25 is divided by 5 first. This gives the intermediate result of 5. Because multiplication also has higher precedence than addition, the intermediate result of 5 is next multiplied by 2, the value of b, giving a new intermediate result of Finally, the addition of 6 and 10 is performed, giving a final result of The second printf statement introduces a new twist. You would expect that dividing a by b and then multiplying by b would return the value of a, which has been set to But this does not seem to be the case, as shown by the output display of It might seem like the computer lost a bit somewhere along the way. The fact of the matter is that this expression was evaluated using integer arithmetic. If you glance back at the declarations for the variables a and b, you will recall that they were both declared to be of type int. Whenever a term to be evaluated in an expression consists of two integers, the C system performs the operation using integer arithmetic. In such a case, all decimal portions of numbers are lost. Therefore, when the value of a is divided by the value of b, or 25 is divided by 2, you get an intermediate result of 12 and not In addition, keep in mind that no rounding occurs, the decimal value is simply dropped, so integer division that ends up with As you can see from the next-to-last printf statement in Program 3. The resulting program is more efficientâ€”that is, it executes more quickly on many computers. On the other hand, if you need the decimal place accuracy, the choice is clear. The only question you then must answer is whether to use a float, double, or long double. The answer to this question depends on the desired accuracy of the numbers you are dealing with, as well as their magnitude. In the last printf statement, the value of the variable a is negated by use of the unary minus operator. A unary operator is one that operates on a single value, as opposed to a binary operator, which operates on two values. The minus sign actually has a dual role: As a binary operator, it is used for subtracting two values; as a unary operator, it is used to negate a value. Once again, in Appendix A you will find a table summarizing the various operators and their precedences. Try to determine how this operator works by analyzing Program 3. For a reminder, before a series of statements that use the modulus operator are printed, the first printf statement prints the values of the four variables used in the program. For the remaining printf lines, as you know, printf uses the character that immediately follows the percent sign to determine how to print the next argument. In the first example, the remainder after 25 is divided by 5 and is displayed as 0. If you divide 25 by 10, you get a remainder of 5, as verified by the second line of output. Dividing 25 by 7 gives a remainder of 4, as shown in the third output line. The last line of output in Program 3. First, you will notice that the program statement has been written on two lines. This is perfectly valid in C. In fact, a program statement can be continued to the next line at any point at which a blank space could be used. The continuation of the printf call in Program 3. Turn your attention to the expression evaluated in the final statement. You will recall that any operations between two integer values in C are performed with integer arithmetic. Therefore, any remainder resulting from the division of two integer values is simply discarded. Multiplying this value by the value of d, which is 7, produces the intermediate result of It is no coincidence that this value is the same as the value of the variable a. As far as precedence is concerned, the modulus operator has equal precedence to the multiplication and division operators. You should note that some compilers might give warning messages to alert you of the fact that conversions are being performed. So, when the value of f1 is assigned to i1 in the previous program, the number Assigning an integer variable to a floating variable does not cause any change in the value of the number; the value is simply converted by the system and stored in the floating variable. The first has to do

with integer arithmetic, which was previously discussed in this chapter. Whenever two operands in an expression are integers and this applies to short, unsigned, long, and long long integers as well , the operation is carried out under the rules of integer arithmetic. Therefore, any decimal portion resulting from a division operation is discarded, even if the result is assigned to a floating variable as you did in the program. Therefore, when the integer variable i2 is divided by the integer constant , the system performs the division as an integer division. The next division performed in the previous listing involves an integer variable and a floating-point constant. Any operation between two values in C is performed as a floating-point operation if either value is a floating-point variable or constant. Therefore, when the value of i2 is divided by The type cast operator has the effect of converting the value of the variable i2 to type float for purposes of evaluation of the expression. In no way does this operator permanently affect the value of the variable i2; it is a unary operator that behaves like other unary operators. The type cast operator has a higher precedence than all the arithmetic operators except the unary minus and unary plus. Of course, if necessary, you can always use parentheses in an expression to force the terms to be evaluated in any desired order. As another example of the use of the type cast operator, the expression int

DATA TYPES, VARIABLES, AND ARITHMETIC pdf

9: www.enganchecubano.com â€“ Relearning MSX # Variables and arithmetic operations in MSX-C (part 2

Chapter 4 introduces variables and arithmetic operations used in the coding of a Visual Basic application. The chapter provides in-depth coverage of declaring variables, gathering input for an application, differentiating data types, performing mathematical calculations, and understanding the proper scope of variables.

Variables A variable is an identifier that points to a memory location where information of a specific data type is stored. The scope of a variable defines the area in the code where an item can be accessed. Instance variables are declared in class declarations, and can be accessed from any methods in the class or from methods that extend the class. Local variables can be accessed only in the block where they were defined. When a variable is declared, memory is allocated, and the variable is initialized to the default value. You can assign values to both static fields and instance fields as part of the declaration statement. Variables can be declared anywhere in a code block in a method. They use the const or readonly keyword. Constants differ from read-only fields in only one way. Read-only fields can be assigned a value only one time, and that value never changes. The field can be assigned its value either inline, at the place where the field is declared, or in the constructor. You can fully qualify the type names in the declaration by including the full namespace, or you can add a using statement to your file and then omit the namespace from the type name. The variable is still strongly-typed into one, unambiguous type. You can use var only on declarations where initialization expressions are provided. The compiler will infer the type from the initialization expression. In some cases, this approach can make code easier to read. You should use var to declare local variables in these situations: Use an explicit type instead. ResultSoFar ; Declare anywhere Declarations can now be provided wherever statements can be provided. A declaration is syntactically a statement, a declaration statement. Therefore, you have precise control over the scope of your variables. Scopes can be started at the block level inside compound statements , in for statements, and in using statements. There are several advantages to making scopes small: You reduce the risk that a variable will be reused in an inappropriate manner during long-term maintenance of the code. In the following example, when the compiler reaches the info statement, it will issue the following error message: You can achieve the same result by putting the object inside a try block and then explicitly calling Dispose in a finally block. In fact, the compiler translates the using statement in just this manner. The following example shows some of the features that we have been describing. For example, the following code will cause the compiler to issue the following diagnostic message: However, constants have the following advantages over macros: You can add a documentation comment to a constant but not to the value of a macro. Eventually, the language service will pick up this comment and provide useful information to the user. A constant is known by IntelliSense. A constant is cross-referenced. Therefore, you can find all references for a specific constant but not for a macro. A constant is subject to access modifiers. You can use the private, protected, and public modifiers. You can see the value of a constant or a read-only variable in the debugger. Macros that are defined in class scopes that is, in class declarations are effectively available in all methods of all derived classes. This feature was originally a bug in the implementation of the legacy compiler macro. However, many application programmers often take advantage of it now. This feature also has a significant effect on the performance of the compiler. Constants can be declared at the class level, as shown in the following example. Therefore, you can easily implement the concept of a macro library. The list of macro symbols becomes a class that has public const definitions. You can also define constants as variables only. You should use variables of this type only as arguments and return values. To use anytype as a variable, you must first assign a value to it. Otherwise, a run-time error occurs. The size, precision, scope, default value, and range of anytype depend on the conversion type that you assign to it. You can use anytype just as you use the data type that you convert it to. For example, if you assign an integer, you can then apply relational and arithmetic operators to the variable. An anytype variable is automatically converted to a date, enumeration enum , integer, real, string, extended data type EDT record , class, or container when a value is assigned to the type. Additionally, the following explicit conversion functions can be used: You can use the reserved literal keywords true and false wherever a boolean expression is expected. The size of a boolean is 1 byte. The

default value is false, and the internal representation is a short number. A boolean is automatically converted to an int, date, or real. It has no explicit conversion functions. The internal representation of a boolean is an integer. You can assign any integer value to a variable that is declared as the boolean type. The integer value 0 zero is evaluated as false, and all other integer values are evaluated as true. Because the internal representation of a boolean is an integer, boolean values are automatically converted to integers and reals. Dates can be written as literals by using the following syntax: You must use four digits for the year. The date data type can hold dates between January 1, , and December 31, The size of a date is 32 bits. The default value is null, and the internal representation is a date. A date has no implicit conversions. However, the following explicit conversion functions can used: If you try, a compiler error occurs. Before you can use an enum, you must declare it in Application Explorer. The literal values are represented internally as integers. The first literal has the number 0, the next literal has the number 1, the next literal has the number 2, and so on. You can use enum values as integers in expressions. The default value for the first entry is 0, and the internal representation is a short number. An enum value is automatically converted to a boolean, int, or real. Hundreds of enumerable types are built into the standard application. For example, the NoYes enum has two associated literals: No has the value 0, and Yes has the value 1. You can create as many enum types as you want, and you can declare up to 0 to literals in a single enum type. To reference an enum value, enter the name of the enum, two colons, and then the name of the literal. Under Artifacts, select Data Types. Click Base Enum to select the new item type. In the Name field, enter a name for the enum, and then click Add. A new enum is added to the project, and the enum designer for the new element is opened. In the enum designer, right-click the enum name, and then click New Element. In the Properties window, enter the name of the enum element. A GUID is an integer that can be used across all computers and networks, wherever a unique identifier is required. A valid GUID meets all the following specifications: It must have 32 hexadecimal digits. It must have four dash characters that are embedded at the following locations: It must have a total of either 36 or 38 characters, depending on whether braces are added. The hexadecimal digits aâ€”f or Aâ€”F can be uppercase, lowercase, or mixed. The size of a guid is 16 bytes or bits. The following explicit conversion functions can be used: The code output of these examples follows. Mixing upper and lower case letters does not affect the guid. Mixed upper and lower case works.

DATA TYPES, VARIABLES, AND ARITHMETIC pdf

Prisoners of war and their captors in World War II The Roots of the Mountain Significance of political science Sothebys guide to American folk art Mastering Excel 3.0 for windows Drawings by Canaletto United states constitutional privileges Plastics Engineered Product Design Vrat katha in gujarati All you need to know about microcomputers Kids answers to lifes big questions Interest groups and lobbying in Latin America, Africa, the Middle East, and Asia The gnu emacs lisp reference manual First age of the Portuguese embassies, navigations, and peregrinations to the kingdoms and islands of Sou International labour conventions and national law 23 11 book 2 Hey bartender give me one more drink World register of production engineering research The history of Hamlet Waffles from morning to midnight The trend of modern poetry. Play piano with diana krall Interfacial science an introduction Apache solr enterprise search server third edition Hyperions Daughters Belly fat burning food list Race and the death penalty Homes and Houses Long Ago (Finding Out About) Determinants of public welfare policies Contextualizing nationalism, transnationalism, and Indian diaspora How to Do Your Own Bricklaying and Stone (Popular Science Skill Book) Ancient Khmer stone carving and bronze casting Pirandello six characters in search of an author Bible readings for couples Subhas chandra bose history in english Life After Deployment Seasons of Communion A report on the banality of evil Guide to the Near Eastern Collections at the University of Pennsylvania Museum of Archaeology and Anthrop Calorie restriction diet plan