Java: Primitive Data Types, Variables and Constants

Introduction

A *primitive data type* is a data type provided as a basic building block by a programming language. It is predefined by the programming language and is named by a reserved keyword or keywords. In Java, primitive data types are used to define *variables* and *constants*. A variable's or constant's data type indicates what sort of value it represents, such as whether it is an integer, a floating-point number or a character, and determines the values it may contain and the operations that may be performed on it.

In Java primitive data types can be used to represent data as *characters*, *integers*, *floating-point numbers* and *boolean values*, which are represented by data types as follows:

character — A *character* is a text character.

- `char` — The `char` data type represents a single character.

integer — An *integer* is a number without a fractional component.

- `byte` — The `byte` data type represents an 8-bit integer.
- `short` — The `short` data type represents a 16-bit integer.
- `int` — The `int` data type represents a 32-bit integer.
- `long` — The `long` data type represents a 64-bit integer.

floating-point number — A *floating-point number* is a real number, or a number that may contain a fractional component. Floating-point types often contain an exponent that represents the number multiplied by 10^x.

- `float` — The `float` data type represents a single-precision floating-point number.
- `double` — The `double` data type represents a double-precision floating-point number.

boolean value — A *boolean value* is a binary value, which can be in one of two states, often *true* or *false*.

- `boolean` — The `boolean` data type represents a boolean value.

This article demonstrates declaration and use of each primitive data type provided by the Java programming language.

The primitive data types available in Java are as follows:

Type	Description	Bytes	Range
`char`	character	2	1 character
`byte`	8-bit integer	1	-128 to 127
`short`	16-bit integer	2	-32,768 to 32,767
`int`	32-bit integer	4	-2,147,483,648 to 2,147,483,647
`long`	64-bit integer	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`float`	single-precision floating-point number	4	$1.4012984632481707*10^{-45}$ to $3.40282346638528860*10^{38}$
`double`	double-precision floating-point number	8	$4.94065645841246544*10^{-324}$ to $1.79769313486231570*10^{308}$
`boolean`	boolean value	1 bit	true or false

The Java programming language is *strongly-typed*, which means that all variables and constants must first be declared before they can be used.

This article demonstrates declaration and use of constants and variables of each primitive data type provided by the Java programming language.

Concepts

value

A *value* is a sequence of bits that is interpreted according to some data type. It is possible for the same sequence of bits to have different values, depending on the type used to interpret its meaning.

In the *primitiv* program, values as literal constants are assigned to variables and constants.

data

Data is a measurement which can be organised to become information.

In English, the word *datum* refers to "something given". The word *data* is plural in English, but it is commonly treated as a mass noun and used in the singular. In everyday language, data is a synonym for information. However, in exact science there is a clear distinction between data and information, where data is a measurement that may be disorganised and when the data becomes organised it becomes information.

The values in the *primitiv* program are data organised according to their *data type*.

bit

The word *bit* is short for *binary digit*, which is the smallest unit of information on a computer. A single bit can hold only one of two values, which are usually represented by numbers 0 and 1.

More meaningful information is obtained by combining consecutive bits into larger units. For example, a *byte* is a unit of information composed of eight consecutive bits.

All values, including those used in the *primitiv* program, are represented by a sequence of bits.

byte

A *byte* is a unit of measurement of information storage, commonly composed of eight bits. If composed of eight bits, a single byte can hold one of 256 (2^8) values.

Data can be represented in a single byte or a combination of bytes.

All values used in the *primitiv* program are represented by a single byte or a combination of bytes.

character

A *character* is a text character.

The *primitiv* program declares character variables of type `char`.

integer

An *integer* is a number without a fractional component.

The *primitiv* program declares integer variables of type `byte`, `short`, `int` and `long`.

floating-point number

A *floating-point number* is a real number, or a number that may contain a fractional component.

The *primitiv* program declares floating-point variables of type `float` and `double`.

boolean value

A *boolean value* is a binary value, which can be in one of two states, often *true* or *false*.

The *primitiv* program declares boolean variables of type `boolean`.

data type

A *data type*, or *type*, is a classification of a particular kind of information. It can be defined by its permissible values and operations.

Data types used in the *primitiv* program are Java primitive data types.

primitive data type

A *primitive data type* is a data type provided as a basic building block by a programming language. It is predefined by the programming language and is named by a reserved keyword.

This article demonstrates declaration and use of each primitive data type provided by the Java programming language.

type specifier

The data type of an entity is specified using a *data type specifier*, sometimes called a *type specifier*.

Data types specified in the *primitiv* program are the Java primitive data types.

identifier

An *identifier* is a token that names an entity.

The concept of an identifier is analogous to that of a name. Naming entities makes it possible to refer to them, which is essential for any kind of symbolic processing.

Variables and constant in the *primitiv* program are represented by identifiers.

keyword

A *keyword* is a word or identifier that has a particular meaning to its programming language.

Keywords are used in the *primitiv* program to specify the types of variables and constant and to declare the constant as a constant.

modifier

A *modifier keyword*, or *modifier*, is a keyword that modifies an entity.

Modifier `final` is used in the *primitiv* program.

declaration statement

A *declaration statement*, often referred to simply as a *declaration*, specifies aspects of an entity, such as its dimensions, identifier and type and is used to announce its existence. This is important in Java which requires variables and constants to be declared before use.

Variables and a constant are declared in the *primitiv* program.

Primitive data types are used in *variables* and *constants*. Java has two kinds of constants: *symbolic constants* and *literal constants*.

variable

A *variable* is a symbolic representation denoting a value or expression that can change. It is a symbol or identifier that stands for a value. For example, in expression

```
b + c
```

`b` and `c` may be variables.

The *primitiv* program uses variables representing each of the primitive data types.

symbolic constant

A *symbolic constant*, *constant variable*, or *constant* for short, is a variable whose value cannot be changed once it is initially bound to a value. A constant variable cannot be assigned to. It is specified only once, but can be referenced multiple times.

The *primitiv* program demonstrates declaration of an integer constant and assignment of a constant value to a variable.

literal constant

A *literal constant*, or *literal*, is a literal value inserted into source code. It is a constant because its value cannot be changed.

The *primitiv* program assigns literal values to variables representing each of the primitive data types.

A *literal constant* literally declares its value, which is directly inserted into the source code of a program. The following kinds of data can be represented by a literal:

character — A single character can be inserted into code by enclosing it in single quotes `''`.

For example, `'a'` represents character `a`.

string — A string of characters can be inserted into code by enclosing the characters in double quotes `""`.

for example, `"Hello"` represents word “Hello”.

integer — An integer can be inserted into code as a number without a decimal point or exponent.

For example, `1234` represents integral number 1234.

If an integer literal is assigned to a variable or constant of type `byte` or `short` and its value is within the valid range, the literal is assumed to be of type `byte` or `short`. An integer literal can have an `l` or `L` suffix that means it is a long integer.

floating-point — A floating-point number can be inserted into code as a number with a decimal point and/or exponent. The exponent is inserted as an integer prefixed with `e` or `E`.

For example, `1.234` represents floating-point number 1.234, `3e2` represents floating-point number 300 and `1.234e2` represents floating-point number 123.4.

By default, a floating-point literal has a type of `double`. To specify a `float` value, the `f` or `F` suffix can be used. A floating-point literal can also have an `d` or `D` suffix to expressly specify that it is of type `double`.

Source Code

The source code listing is as follows:

```
/*
  primitiv.java

  Primitive data types.

  environment: language Java
               platform console
*/

class primitiv
{
  public static void main( String[] args )
  {
    // Declare variables.

    char    c                 ; // character
    int     in , iz , ip , ic ; // integer
    float   fn , fz , fp , ff ; // floating-point
    boolean bf , bt           ; // boolean

    // floating-point

    float fm6 , fm5 , fm4 , fm3 , fm2 , fm1 ,
          f0  , f1  , f2  , f3  , f4  , f5  , f6 , f7 ;

    // range: integer

    byte  scn , scp ;
    short ssn , ssp ;
    int   sin , sip ;
    long  sln , slp ;

    // range: floating-point

    float  fns , fnl , fps , fpl ;
    double dns , dnl , dps , dpl ;

    // Populate variables.

    // character

         c = 'c' ;
    char p = 'p' ;
```

```
    // integer

    final int cic =  345 ;

              in  = -123 ;
              iz  =    0 ;
              ip  =  123 ;
              ic  = cic  ;

    // floating-point

    fn  = -123       ;
    fz  =    0       ;
    fp  =  123       ;
    ff  =   12.3f    ;

    fm6 =    1.23e-6f ;
    fm5 =    1.23e-5f ;
    fm4 =    1.23e-4f ;
    fm3 =    1.23e-3f ;
    fm2 =    1.23e-2f ;
    fm1 =    1.23e-1f ;
    f0  =    1.23e0f  ;
    f1  =    1.23e1f  ;
    f2  =    1.23e2f  ;
    f3  =    1.23e3f  ;
    f4  =    1.23e4f  ;
    f5  =    1.23e5f  ;
    f6  =    1.23e6f  ;
    f7  =    1.23e7f  ;

    // boolean

    bf = false ;
    bt = true  ;

    // range: integer

    scn  =                -128  ;
    scp  =                 127  ;

    ssn  =              -32768  ;
    ssp  =               32767  ;

    sin  =         -2147483648  ;
    sip  =          2147483647  ;

    sln  = -9223372036854775808l ;
    slp  =  9223372036854775807l ;
```

```
    // range: floating-point

    fns  = -1.40129846432481707e-45f ;
    fnl  = -3.40282346638528860e+38f ;
    fps  =  1.40129846432481707e-45f ;
    fpl  =  3.40282346638528860e+38f ;

    dns  = -4.94065645841246544e-324 ;
    dnl  = -1.79769313486231570e+308 ;
    dps  =  4.94065645841246544e-324 ;
    dpl  =  1.79769313486231570e+308 ;

    // Display variables.

    System.out.print( "char     " ) ;
    System.out.print( c )          ;
    System.out.print( " " )        ;
    System.out.println( p )        ;

    System.out.print( "int      " ) ;
    System.out.print( in )         ;
    System.out.print( " " )        ;
    System.out.print( iz )         ;
    System.out.print( " " )        ;
    System.out.println( ip )       ;

    System.out.print( "float    " ) ;
    System.out.print( fn )         ;
    System.out.print( " " )        ;
    System.out.print( fz )         ;
    System.out.print( " " )        ;
    System.out.print( fp )         ;
    System.out.print( " " )        ;
    System.out.println( ff )       ;
    System.out.print( "         " ) ;
    System.out.print( fm6 )        ;
    System.out.print( " " )        ;
    System.out.print( fm5 )        ;
    System.out.print( " " )        ;
    System.out.print( fm4 )        ;
    System.out.print( " " )        ;
    System.out.print( fm3 )        ;
    System.out.print( " " )        ;
    System.out.print( fm2 )        ;
    System.out.print( " " )        ;
    System.out.println( fm1 )      ;
    System.out.print( "         " ) ;
    System.out.print( f0 )         ;
    System.out.print( " " )        ;
    System.out.print( f1 )         ;
```

```
    System.out.print( " " )          ;
    System.out.print( f2 )           ;
    System.out.print( " " )          ;
    System.out.print( f3 )           ;
    System.out.print( " " )          ;
    System.out.print( f4 )           ;
    System.out.print( " " )          ;
    System.out.print( f5 )           ;
    System.out.print( " " )          ;
    System.out.print( f6 )           ;
    System.out.print( " " )          ;
    System.out.println( f7 )         ;

    System.out.print( "bool     " ) ;
    System.out.print( bf )           ;
    System.out.print( " " )          ;
    System.out.println( bt )         ;

    System.out.print( "constant " ) ;
    System.out.println( ic )         ;

    // range

    System.out.println()             ;
    System.out.println( "range:" ) ;
    System.out.println()             ;

    System.out.print( "byte   " )  ;
    System.out.print( scn )         ;
    System.out.print( " " )         ;
    System.out.println( scp )       ;

    System.out.print( "short  " )  ;
    System.out.print( ssn )         ;
    System.out.print( " " )         ;
    System.out.println( ssp )       ;

    System.out.print( "int    " )  ;
    System.out.print( sin )         ;
    System.out.print( " " )         ;
    System.out.println( sip )       ;

    System.out.print( "long   " )  ;
    System.out.print( sln )         ;
    System.out.print( " " )         ;
    System.out.println( slp )       ;

    System.out.print( "float  " )  ;
    System.out.print( fns )         ;
    System.out.print( " " )         ;
```

```
    System.out.print( fnl )         ;
    System.out.print( " " )         ;
    System.out.print( fps )         ;
    System.out.print( " " )         ;
    System.out.println( fpl )       ;

    System.out.print( "double " )   ;
    System.out.print( dns )         ;
    System.out.print( " " )         ;
    System.out.print( dnl )         ;
    System.out.print( " " )         ;
    System.out.print( dps )         ;
    System.out.print( " " )         ;
    System.out.println( dpl )       ;
  }
}
```

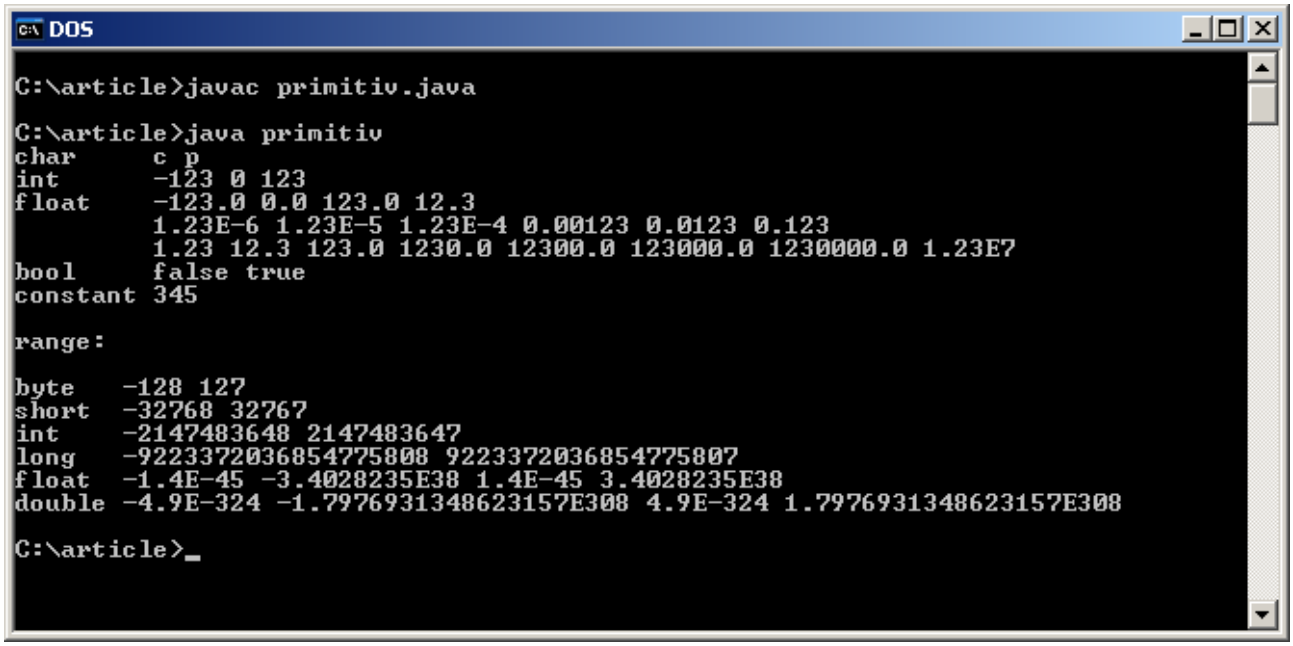
Compiling and Running

1. Save the source code listing into a file named `primitiv.java`.
2. Launch a Windows command prompt.
3. Navigate to the directory `primitiv.java` was saved in.
4. To compile the program, type:

   ```
   > javac primitiv.java
   ```

5. To run the program, type

   ```
   > java primitiv
   ```



Code Explanation

```
char c ; // character
```

A variable is a symbolic representation denoting a value or expression. The format of a simple variable declaration is as follows:

```
type identifier ;
```

where

`type` is the type specifier of the variable.

The type specifier in the declaration statement above is `char`, which specifies a character.

`identifier` is the identifier by which it will be possible to refer to the variable.

The identifier in the declaration statement above is `c`.

`;` closes the declaration and delimits it from the next statement.

All statements in Java, including the declaration statement above, are closed with the semicolon character `;`.

The statement above declares a character variable identified by `c`.

```
int in , iz , ip , ic ; // integer
```

Multiple variables of the same type can be declared in a single declaration statement. The format of a multiple variable declaration is as follows:

```
type identifier[ , identifier[ , identifier[ , ... ]]] ;
```

where

`type` is the type specifier of the variables.

The type specifier in the declaration statement above is `int`, which specifies integers.

`identifier` is the identifier by which it will be possible to refer to the variable.

The identifiers in the declaration statement above are `in`, `iz`, `ip` and `ic`.

`,` delimits the variables being identified.

All multiple declarations in Java, including the declaration above, delimit the variables with commas `,`.

`;` closes the declaration and delimits it from the next statement.

All statements in Java, including the declaration statement above, are closed with the semicolon character `;`.

The statement above declares multiple integer variables identified by `in`, `iz`, `ip` and `ic`.

```
float   fn , fz , fp , ff ; // floating-point
boolean bf , bt           ; // boolean
```

These statements declare floating-point and boolean variables.

```
float fm6 , fm5 , fm4 , fm3 , fm2 , fm1 ,
      f0  , f1  , f2  , f3  , f4  , f5  , f6 , f7 ;
```

The floating-point variables declared in this statement are used to demonstrate a range of floating-point values.

```
byte  scn , scp ;
short ssn , ssp ;
int   sin , sip ;
long  sln , slp ;
```

These statements declare variables for each of the primitive integral types provided by the Java programming language.

```
float  fns , fnl , fps , fpl ;
double dns , dnl , dps , dpl ;
```

These statements declare variables for each of the primitive floating-point types provided by the Java programming language.

```
c = 'c' ;
```

A character variable stores a single character.

To assign is to set or re-set a value denoted by an identifier. An assignment statement uses the assignment operator = to assign the value of an expression to an entity. It is formatted as follows:

```
assignee = expression ;
```

where

`assignee` is the entity (variable or constant) to which the value is assigned.

The assignee in the statement above is variable `c`.

`=` is the assignment operator.

The assignment operator in the statement above, as in all assignment statements is denoted by the equal character =.

`expression` is the expression that is evaluated.

The expression in the statement above is literal constant `'c'`.

`;` closes the statement and delimits it from the next statement.

All statements in Java, including the statement above, are closed with the semicolon character `;`.

The statement above assigns literal value “c” to character variable `c`.

```
char p = 'p' ;
```

A declaration and assignment can be combined into a single statement. A combined declaration and assignment statement contains the elements of both a declaration statement and an assignment statement, and is formatted as follows:

```
type identifier = expression ;
```

where

`type` is the type specifier of the entity being declared.

The type specifier in the statement above is `char`, which is used to specify a character.

`identifier` is the identifier by which it will be possible to refer to the entity.

The identifier in the statement above is `p`.

`=` is the assignment operator.

The assignment operator in the statement above, as in all assignment statements is denoted by the equal character =.

`expression` is the expression that is evaluated.

The expression in the statement above is literal constant `'p'`.

`;` closes the statement and delimits it from the next statement.

All statements in Java, including the statement above, are closed with the semicolon character `;`.

The statement above declares a character variable identified by `p` and assigns literal value “p” to it.

```
final int cic = 345 ;
```

A constant can be declared in the same way as a variable, preceded by the `final` modifier. The `final` modifier defines an entity that cannot later be changed.

This statement declares an integer constant identified by `cic` and assigns literal value 345 to it.

```
in = -123 ;
iz =    0 ;
ip =  123 ;
```

An integer can store positive and negative integral values.

These statements assign literal values -123, 0 and 123 to integer variables `in`, `iz` and `ip` respectively.

```
ic = cic ;
```

Valid expressions in an assignment statement include literal and symbolic constants and variables.

This statement assigns the value of integer symbolic constant `cic` to integer variable `ic`.

```
fn = -123    ;
fz =    0    ;
fp =  123    ;
ff =   12.3f ;
```

A floating-point number can store positive and negative values and may also contain a fractional component.

These statements assign literal values -123, 0, 123 and 12.3 to floating-point variables `fn`, `fz`, `fp` and `ff` respectively.

By default a floating-point literal is of type `double`. Assignment statement `ff = 12.3 ;` would therefore cause the compiler to return error “possible loss of precision”. By using the `f` suffix, the type of the floating-point literal can be specified as `float`, and the above statement `ff = 12.3f ;` assigns the value without causing a compiler error.

```
fm6 = 1.23e-6f ;
fm5 = 1.23e-5f ;
fm4 = 1.23e-4f ;
fm3 = 1.23e-3f ;
fm2 = 1.23e-2f ;
fm1 = 1.23e-1f ;
f0  = 1.23e0f  ;
f1  = 1.23e1f  ;
f2  = 1.23e2f  ;
f3  = 1.23e3f  ;
f4  = 1.23e4f  ;
f5  = 1.23e5f  ;
f6  = 1.23e6f  ;
f7  = 1.23e7f  ;
```

Floating-point types can contain an exponent that represents the number multiplied by $10^{<x>}$. This is represented by appending `e<x>` to the number where `<x>` represents a power of 10.

These statements assign literal values to floating-point variables as follows:

$1.23 * 10^{-6}$	`fm6`
$1.23 * 10^{-5}$	`fm5`
$1.23 * 10^{-4}$	`fm4`
$1.23 * 10^{-3}$	`fm3`
$1.23 * 10^{-2}$	`fm2`
$1.23 * 10^{-1}$	`fm1`
$1.23 * 10^{0}$	`f0`
$1.23 * 10^{1}$	`f1`
$1.23 * 10^{2}$	`f2`
$1.23 * 10^{3}$	`f3`
$1.23 * 10^{4}$	`f4`
$1.23 * 10^{5}$	`f5`
$1.23 * 10^{6}$	`f6`
$1.23 * 10^{7}$	`f7`

When the *primitiv* program runs these variables are displayed as `1.23E-6`, `1.23E-5`, `1.23E-4`, `0.00123`, `0.0123`, `0.123`, `1.23`, `12.3`, `123.0`, `1230.0`, `12300.0`, `123000.0`, `1230000.0` and `1.23E7` respectively. Each number is displayed with or without the exponent as appropriate to make the number readable.

```
bf = false ;
bt = true  ;
```

A boolean value is a binary value, or one that can be in one of two states, often true or false.

These statements assign literal values false and true to boolean variables `bf` and `bt` respectively.

```
scn =                  -128  ;
scp =                   127  ;

ssn =                -32768  ;
ssp =                 32767  ;

sin =           -2147483648  ;
sip =            2147483647  ;

sln = -92233720368547758081 ;
slp =  92233720368547758071 ;
```

The range of values that can be contained in each integer type is as follows:

`byte` -128 to 127

`short` -32,768 to 32,767

`int` -2,147,483,648 to 2,147,483,647

`long` -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

The statements above assign the minimum and maximum literal values to variables of each of the integer types.

If an integer literal is assigned to a variable of type `byte` or `short` and its value is within the valid range, the literal is assumed to be of type `byte` or `short`. To assign an integer literal to a variable of type `long`, the `l` suffix is required. Assignment statement `sln = -92233720368547758081 ;` would cause the compiler to return error "integer number too large". By using the `l` suffix, the type of the integer literal can be specified as `long`, and the above statement `sln = -92233720368547758081 ;` assigns the value without causing a compiler error.

```
fns = -1.40129846432481707e-45f ;
fnl = -3.40282346638528860e+38f ;
fps =  1.40129846432481707e-45f ;
fpl =  3.40282346638528860e+38f ;

dns = -4.94065645841246544e-324 ;
dnl = -1.79769313486231570e+308 ;
dps =  4.94065645841246544e-324 ;
dpl =  1.79769313486231570e+308 ;
```

The range of absolute values that can be contained in each floating-point type is as follows:

`float`	$1.40129846432481707*10^{-45}$ to $3.40282346638528860*10^{38}$
`double`	$4.94065645841246544*10^{-324}$ to $1.79769313486231570*10^{308}$

The statements above assign the minimum and maximum positive and negative literal values to variables of each of the floating-point types.

```
System.out.print( "char     " ) ;
```

The `print` method of the `out` property of the `System` class can be used to write a line of data to the standard output stream. This method is the same as the `println` method except that it does not terminate the line it printed to.

This statement uses the `print` method to print string literal “char ”.

```
System.out.print( c ) ;
```

The `print` and `println` methods can write variables and literal and symbolic constants to the standard output stream.

This statement writes the value of character variable `c` to the standard output stream.

Terms

expression

An *expression* is a programming language construct that evaluates to some quantity.

For example, `b + c` is an expression that evaluates to the sum of `b` plus `c`. If `b = 2` and `c = 3`, expression `b + c` evaluates to `5 (2 + 3 = 5)`.

Each expression in the *primitiv* program consists of a single variable, constant or literal.

assign

To *assign* is to set or re-set a value denoted by a variable name.

Values are assigned to each of the variables and constants declared in the *primitiv* program.

assignment statement

An *assignment statement* assigns a value to a variable. The assignment statement often allows a variable to contain different values at different times during program execution.

Values in the *primitiv* program are assigned using assignment statements.

assignment operator

An assignment statement uses the *assignment operator* = to assign the value of an expression to an entity.

Assignment statements in the *primitiv* program use the assignment operator.

Further Comments

Declarations

The declaration formats provided earlier in this article were simplified to explain their use in context. A more complete outline of the format of a variable or constant declaration is as follows:

```
[final ]type
identifier[ = expression][ , identifier[ = expression][ , ...]]
;
```

where

`final`	is the `final` keyword and indicates that the variable cannot later be changed. This effectively declares the variable as a constant. If the `final` modifier is omitted, a variable is declared.
`type`	is the type specifier of the variable or constant.
`identifier`	is the identifier by which it will be possible to refer to the variable or constant.
`=`	is the assignment operator = and indicates that the value of an expression is to be assigned.
`expression`	is the expression that is evaluated.
`,`	delimits the variables or constants being identified.
`;`	closes the statement and delimits it from the next statement.

Characters

The `char` data type is a single 16-bit Unicode character.

Unicode is an encoding standard in which all characters are two bytes long. Literals of type `char` may contain any Unicode (UTF-16) character.