CMP 1001

Introduction to Programming

Part 2: Variables & Basic Operators in C++

by

Assist. Prof. M. Şükrü Kuran

CMP 1001

Section 3: Variables & Constants

VARIABLES

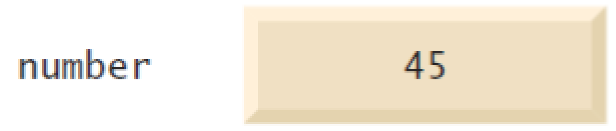
What is a variable?

Variables are extremely important and key elements to any program.

They define memory devices or storage places to keep “**values**” in the program.

```
<variable type> <variable name>;
```

```
int number;
number = 45;
```



```
int number = 0;
```

NOTE: When you declare a variable, you SHOULD initialize them (i.e., give them their first values.)

VARIABLES

Features of Variables

1. **Variable Type**

- When we declare a variable, we clearly specify its type (e.g., integer, double, char).
- The type of a variable do **NOT** change through the program.

2. **Variable Name**

- A variable has a well-defined name. Variable names do **NOT** change through the program.
- **NO TWO** variables can have the same name (unless scopes are involved).

3. **Value**

- The value that the variable stores inside.
- The value of the variable **CAN** change over time based on the program.

VARIABLE NAMING

Rules for Variable Naming

1. They must consist of **LETTERS & DIGITS.** Also, the underscore character '_' and dollar sign '$' can be used.

   NOTE: The '_' and '$' characters usually have special uses, so do NOT use them.

2. You **CANNOT** use non-English (i.e., Turkish) characters in variable names.

   NOTE: As a general programming habit, ONLY use English words.

3. The names of variables **MUST** begin with letters. It is also possible to begin with '_' or '$', but it is not recommended since library routines often use such names.
4. C++ is a case sensitive language (i.e., **"x" AND "X" ARE DIFFERENT**).
5. Keywords like int, main, float, return, long, if, etc. are reserved and **CANNOT** be used as identifier names.
6. *The language does **NOT** require that variable names are meaningful, but it is **STRONGLY** recommended that you use meaningful names.*

VARIABLE NAMING

Examples

INVALID variable names	**VALID** variable names
8Name_4	My_Name8
4564	number_
number+	_number
data\%	COMP
x9875424	fdjkfhdg34ffs
double	x9875424

NOTE: These rules ALSO apply to other identifier names such as function names, class names, …

“const” - CONSTANTS

Fix valued variables

There are identifiers defined as constants. You are **NOT** allowed to change their values during the execution of the program.

```
const double PI = 3.14;
```

By convention, only **CAPITAL LETTERS** and the ‘_’ character are used within constant names.

Example:
Write down a program which reads the radius of a circle and then calculates the circumference and area of the circle using the PI constant shown above.

CMP 1001

Section 4: Operators

OPERATORS

Various types of operators in C++

Assignment operator	=								
Arithmetic operators	+	-	*	/	%				
Compound operators	+=	-=	*=	/=	%=	>>=	<<=	^=	|=
Inc/Dec operators	++	--							
Comparison operators	==	!=	<	>	<=	>=			
Logical operators	!	&&	||						
Bitwise operators	&	|	^	~	<<	>>			
Others…	,	.	.*	->	->*	()	[]	new	delete

ASSIGNMENT OPERATOR

Core Mathematical Operator

"**Assignment operator**" ' = ' is one of the **MOST COMMON** operators.

```
<left value> = <right value>;
```

' = ' calculates the value of what is on the **RIGHT HAND SIDE** of the statement and assigns this value to the variable on the **LEFT HAND SIDE** of the statement.

```
x = 5;
```

```
x = 6 + a + 2;
```

VALUE CATEGORIES

C++11 Complex Value Categories

$$\begin{tikzcd}
 & & \text{expression} \arrow[dl] \arrow[dr] & & \\
 & \text{glvalue} \arrow[dl] \arrow[dr] & & \text{rvalue} \arrow[dl] \arrow[dr] & \\
\text{lvalue} & & \text{xvalue} & & \text{prvalue}
\end{tikzcd}$$

Value Categories in cppreference.com

https://en.cppreference.com/w/cpp/language/value_category

WORKING WITH DIFFERENT TYPES

Each variable type has a size in terms of bits.

```
int integerNumber = 5;
double doubleNumber = 7.67;
```

```
doubleNumber = integerNumber;
```

```
integerNumber = doubleNumber;
```

Promotion

Type Casting

WORKING WITH DIFFERENT TYPES

"**Promotion**": When a narrower (i.e., smaller) fundamental type is assigned to a wider (i.e., bigger) one, the value is assigned without losing information automatically.

"**Type Casting**": When a wider type is assigned to a narrower one, it is type casted implicitly. However, it is HIGHLY RECOMMENDED that you should use the appropriate "**casting operator**".

NOTE: C++ DOES consider bool variables as numbers. Therefore, in C++, bool variables CAN BE cast into other fundamental types.

PROMOTION

Valid promotions between fundamental types:

Source Fundamental Type	Valid Promotable Fundamental Type
long double	-
double	long double
float	long double, double
long long int	long double, double
long int	long double, double, long long int
int	long double, double, float, long long int, long int
char	long double, double, float, long long int, long int, int, short int
short int	long double, double, float, long long int, long int, int, char
boolean	long double, double, float, long long int, long int, int, short int, char

TYPE CASTING

Two types of type casting:

“**Implicit**”: When you simply put a wider value into a narrower one. C++ automatically casts the wider value into the narrower value.

“**Explicit**”: Instead, you use an explicit type casting operator.

NOTE: You should avoid implicit type casting since how it will work depends on the compiler.

CASTING OPERATORS

Type conversion

Explicit type conversion can be enforced by “**casting operators**”.

Target Type	Casting Operator
long double	(long double)
double	(double)
float	(float)
bool	(bool)

Target Type	Casting Operator
long long int	(long long int)
long int	(long int)
int	(int)
char	(char)
short int	(short int)

PROMOTION AND TYPE CASTING

Example:

Write a program that declares one int and one double variable, initializes them with a valid input (i.e., an integer value for the int and a real value for the double), then

- **Copies the int's value into the double variable**
- **Puts a real number into the int variable**

ARITHMETIC OPERATORS

Classical mathematical operators

"**Arithmetic operators**", '+', '-', '*', '/', and '%' are classical mathematical operators.

These can be applied to **ALL** fundamental types **INCLUDING** bool.

NOTE: When the left hand side of an operator has a narrower type than the right hand side one, the narrower one is promoted to the wider one.

When we use the '/' division operator, there are two possible results:

1. **Integer division**: If both operands are integers. Truncates any fraction part.
2. **Real division**: If **AT LEAST** one operand is float, double or long double. Also calculates the fraction part.

ARITHMETIC OPERATORS

Integer and Real Divisions

Example:
Write a program that reads two numbers as integers, calculates the division of them via
- **Integer division**
- **Real division**

Example:
Check double – int interaction in various arithmetic operators.

Example:
Check char – int and char – double interactions in various arithmetic operators.

COMPOUND OPERATORS

Combined arithmetic and assignment operators

Generally used to shorten the code.

```
    x += y; //shorthand for x = x + y;
x *= y + 1; //shorthand for x = x * (y+1);
    x -= 5; //shorthand for x = x – 5;
x /= y + 2; //shorthand for x = x / (y+2);
```

INCREMENT/DECREMENT

Shorthand for simple addition or subtraction

“++” and “--“ are the “**increment and the decrement operators**” respectively.

“++” adds 1 to its operand. “--” subtracts 1 from its operand.

```
m = n++; // Assign the value of n to m, then increment n’s value by 1
m = n--; // Assign the value of n to m, then decrement n’s value by 1
```

```
m = ++n; // Increment n’s value by 1, then assign the value of n to m
m = --n; // Decrement n’s value by 1, then assign the value of n to m
```

Example:
Check both of the increment operator uses (i.e., postfix and prefix versions).

COMMENT LINES

Self-notes of the programmer

We can write some notes to ourselves inside programs.

These notes are little descriptions of the program.

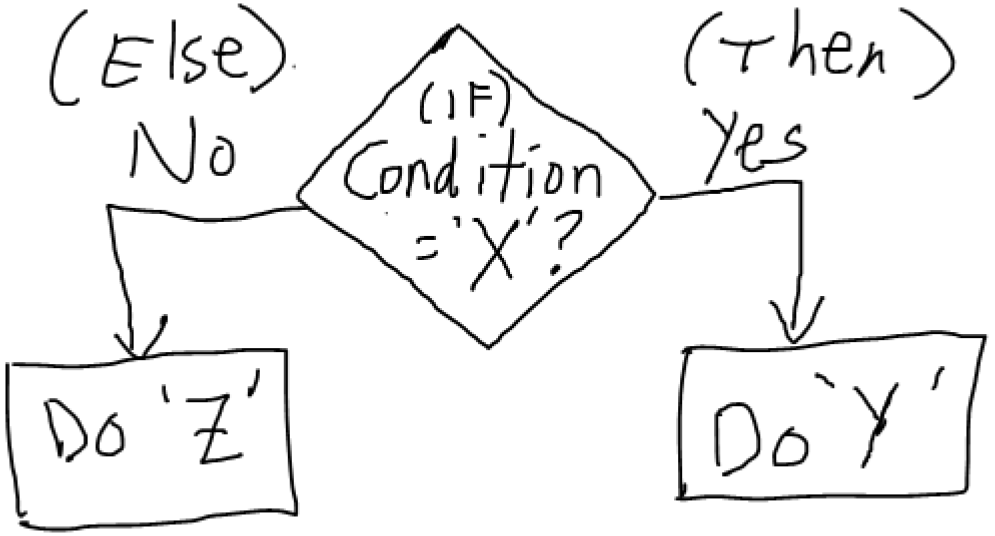
These lines are called “**comments**” and they are **NOT** statements.

- Line starts with “//” is considered to be a comment.
- Lines between “/*” and “*/” are considered to be comments.

C++ does **NOT CARE** and **USE** anything in these comment.

COMING SOON...

Next week on CMP 1001



Conditional Statements in C++