Chapter - 4

Basic Declarations and Expressions

Elements of a Program

- Data

  The basic building blocks of a program.

- Code

  Code tells the computer what to do with the blocks. For example, assemble them into a building.

- Classes (discussed later)

  A combination of data and the operations that be performed on it.

```
#include <iostream>
 int height;  // Height of a rectangle (in inches)
 int width;   // Width of the rectangle (in inches)
 int area;    // Area of the rectangle (in square
   inches)

 int main(){
     height = 3;
     width = 5
     area = height * width;
     std::cout << "Area is " << area << " sq. inches\n";
     return (0);
 }
```

Basic Program Structure

```
/*****************************************
                                         *
 *****************************************/

main()
{


}
```

Hello World

```
/********************************************************
 * hello -- program to print out "Hello World".         *
 *      Not an especially earth-shattering program.     *
 *                                                      *
 * Author: Steve Oualline                               *
 *                                                      *
 * Purpose: Demonstration of a simple program           *
 *                                                      *
 * Usage:                                               *
 *      Run the program and the message appears         *
 ********************************************************/
#include <iostream>
int main()
{
    // Tell the world hello
    std::cout << "Hello World\n";
    return (0);
}
```

Simple Expressions

Operator	**Meaning**
`*`	Multiply
`/`	Divide
`+`	Add
`-`	Subtract
`%`	Modulus (remainder after division)

Precedence Rules

Multiply (*), divide (/) and modulus (%) have precedence over addition (+) and subtraction (-). Parentheses may be used to group terms. Thus:

```
(1 + 2) * 4
```

yields 12, while:

```
1 + 2 * 4
```

yields 9.

Using Expressions

```
int main()
{


}
```

There is a problem with this program. We compute the value of the expressions, but we don’t do anything with it.

If we were constructing a building, think about how confused a workman would be if we said,

“Take your wheelbarrow and go back and forth between the truck and the building site.”
“Do you want me to carry bricks in it?”
“No. Just go back and forth.”

Using the `std::cout` output class

```
std::cout << "Hello World\n";
std::cout << "Hello " << "World\n";
std::cout << "Half of " << 64 << " is " << (64 / 2) << "\n";
```

Outputs

```
Half of 64 is 32
```

Watch the spacing:

```
// Problem code
std::cout << "Half of" << 64 << "is" <<
             (64 / 2) << "\n";
```

Outputs

```
Half of64is32
```

Variables and Storage

- Each variable must be named
- Variables are case sensitive. The names “sam,” “Sam,” and “SAM” specify three different variables.
- By convention variable names are lower case only.

The following is an example of some variable names:

```
average     // average of all grades
pi          // pi to 6 decimal places
number_of_students // # students in this class
```

More on Names

The following are *not* variable names:

```
3rd_entry   // Begins with a number
all$done    // Contains a "$"
the end     // Contains a space
int         // Reserved word
```

Avoid variable names that are similar. For example the following illustrates a poor choice of variable names:

```
total       // total number of items in current entry
totals      // total of all entries
```

A much better set of names is:

```
entry_total // total number of items in current entry
all_total   // total of all entries
```

Variable Declarations

A variable declaration serves three purposes:

1. It defines the name of the variable.

2. It defines the type of the variable (integer, real, character, etc.).

3. It gives the programmer a description of the variable.

Declaration format:

```
     type  name;   // comment
```

Example:

```
    int answer;    // the result of
                   // our expression
```

Integers

Integers: 1, 87, -222, 32767
Not integers. 8.3, 4.8, -5.6
Integer declarations

```
     int  name;   // comment
```

Integer Limitations

On an 8 digit calculator you can only use the numbers 99999999 to -99999999.
Integers have a similar limitation.

On *most* machines integers are 32 bits (4 bytes), providing a range of 2147483647 (2^{31}-1) to -2147483648(-2^{31}). On some older machines integers are only 16 bits (2 bytes) are used, so the range is 32767 (2^{15}-1) to -32768(-2^{15}). Finally, new 64 bit machines are coming which have 4 byte integers.

Question:

Why does the following code work just find on a UNIX machine, but fail on a PC?

```
// zip code for current address
int zip;

.........

zip = 92126;
```

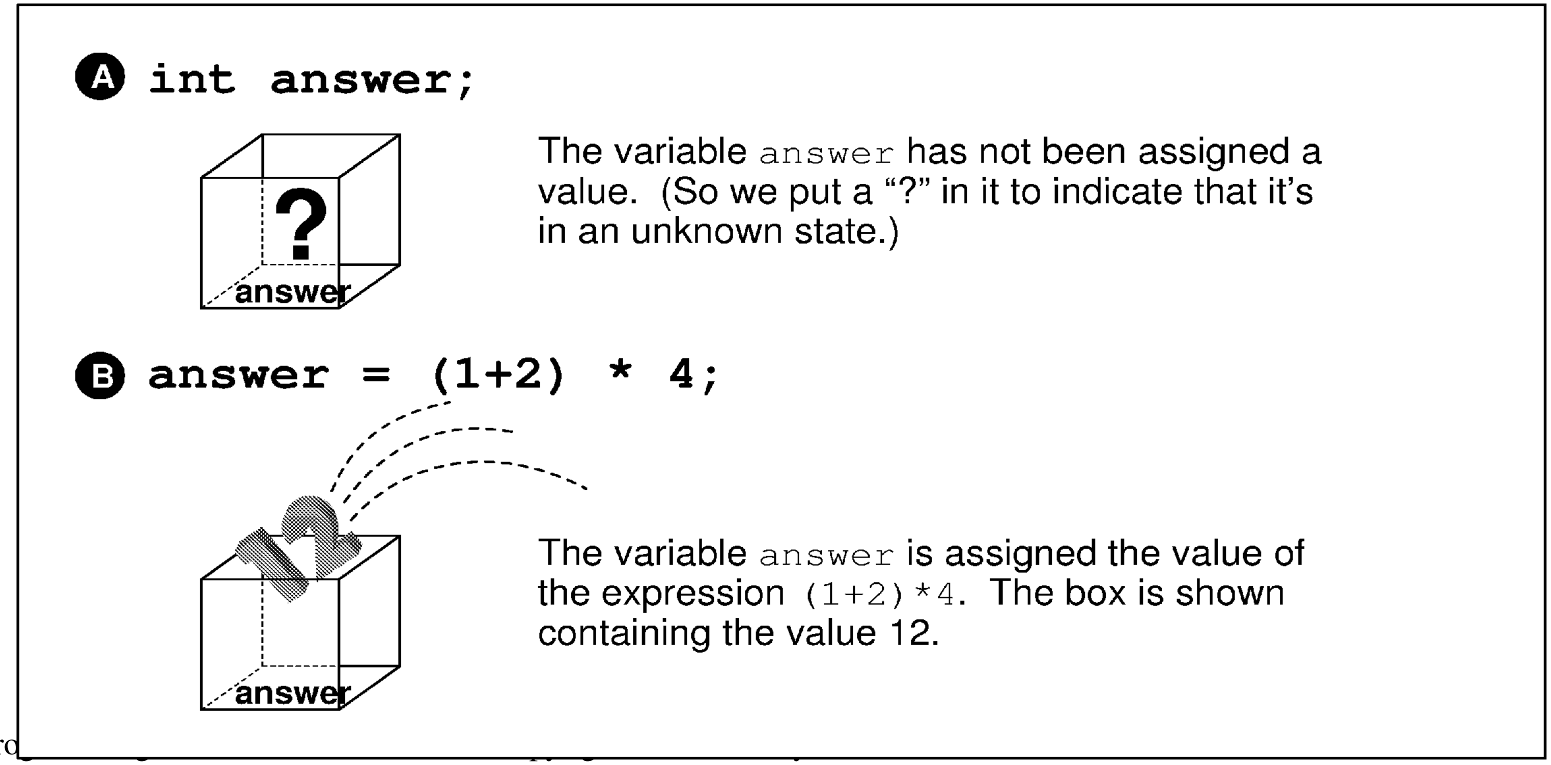
Assignment Statements

Once a variable is declared:

```
   int answer;      // A place to put our results
```

it can be used in an assignment statement:

```
   answer = (1 + 2) * 4;
```



```
int answer;
```

The variable `answer` has not been assigned a value. (So we put a “?” in it to indicate that it’s in an unknown state.)

```
answer = (1+2) * 4;
```

The variable `answer` is assigned the value of the expression `(1+2)*4`. The box is shown containing the value 12.

Assignment Example

```
int main()
{




}
```

Floating Point

Floating point numbers, also called real numbers, contain a decimal point.

Examples: 5.5, 8.3, -12.6.

Note: 5.0 is floating point. 5 is integer.

It is possible to omit the leading or trailing zero in floating point numbers.

For example 5. is the same as 5.0 and .2 is the same as 0.2. However adding the extra zero makes obvious that you are using a floating point number.

Example “5.” (floating point) looks like “5” (integer). “5.0” does not look like “5”.

Exponent notation for floating point numbers: *e±exp*

For example, 1.2e34, is shorthand for $1.2*10^{34}$.

Floating Point Declarations

```
float   variable;   // comment
```

Writing floating point numbers:

```
float a_result; // A place to put the result

int main()
{
    a_result = 1.0 / 3.0;
    std::cout << "One-third is " <<
                     a_result << '\n';
    return (0);
}
```

Floating Point vs. Integer

Integer divide truncates so 19/10 is 1.

Floating point divide does not 19.0 / 10.0 is 1.9.

A divide is floating point if either operand is a floating point number.

Expression	Result	Result Type
`19/10`	`1`	Integer
`19.0/10`	`1.9`	Floating point
`19/10.0`	`1.9`	Floating point
`19.0/10.0`	`1.9`	Floating point

Integers may be assigned to floating point numbers.

Floating point numbers may be assigned to integers (the floating point number is truncated.)

```
    i = 1.9;     // i is assigned the value 1
```

Example

```
int main()
{








}
```

[illegible]

Question

Why is the result of this program “0”? What must be done to fix it?

```
int main()
{




}
```

Character Variables

Declaration

```
char   variable;   //comment
```

Characters are enclosed in single quotes (‘) . Examples ‘A’, ‘a’ and ‘! ’.

The backslash(\) is used to indicate special characters.

Example ‘\n’ is the newline character. (Advances the output to the next line.)

Example

```
int main(){




                    "\n";


}
```

Wide Character Variables

Used for foreign languages that can't be represented using simple characters (Chinese, Japanese, Farsi, Hebrew, etc.)

Declaration

```
        w_char   variable;   //comment
```

Wide characters constants have the form `L'char' (:L" is for "long")`

Example:

```
        w_char   center;   // A Chinese char
        center = L'?  ';
```

Boolean (`bool`) type

Boolean varaible can have one of two values:

```
true
false
```

Example:

```
bool flag;
flag = true;
```

Note: The `bool` type is relatively new to C++ and some legacy macros exist to implement a bool type. These macros use `BOOL` or `Bool` as a data type and `TRUE` and `FALSE` as the values. (These legacy types should be avoided.)