Arithmetic rules:

Operand types	Result type
int & int	int
real & int	real
real & real	real

Type conversion (casting)

type (variable);

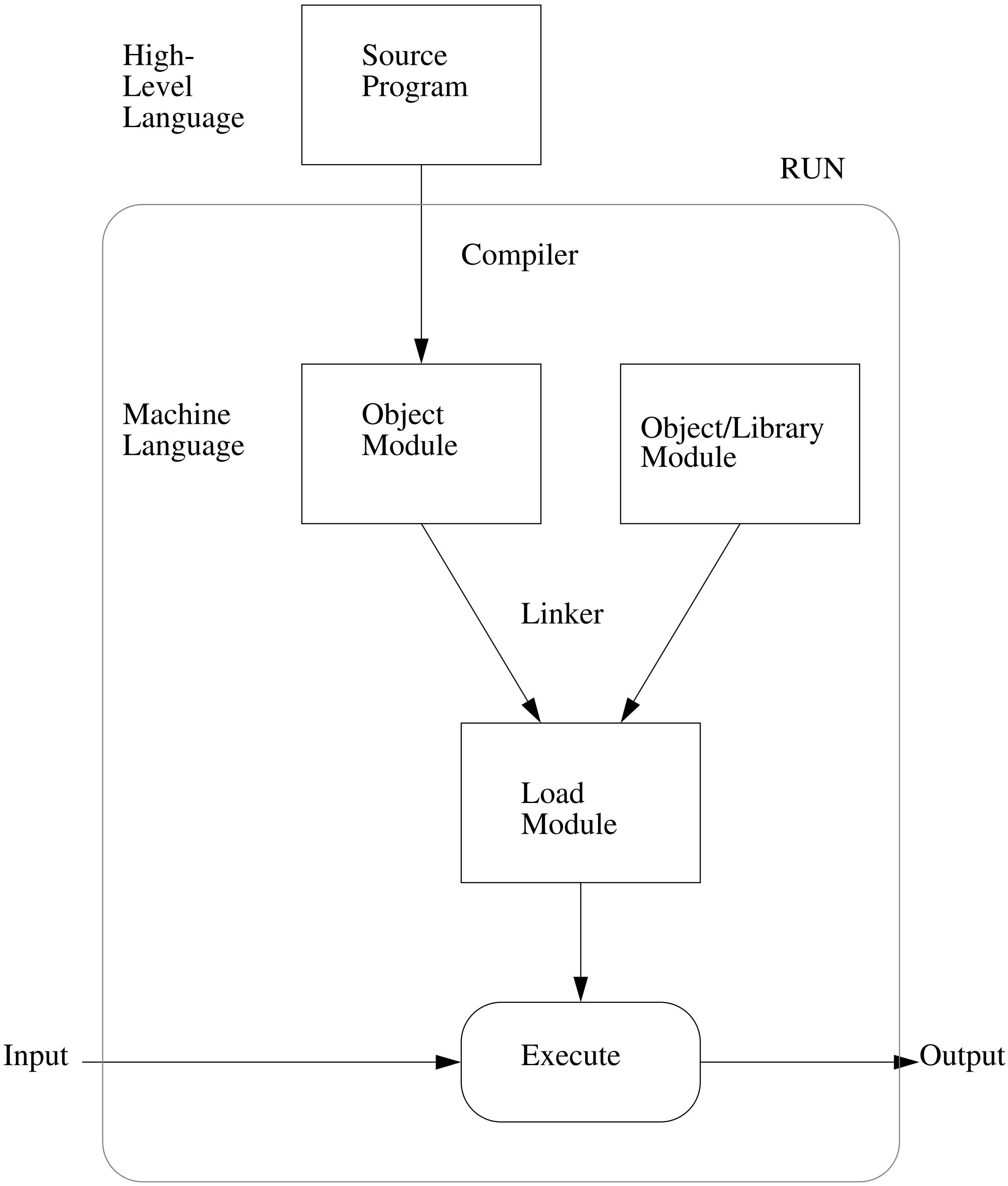
type (expression);

e.g.

float (int_var1) / int_var2

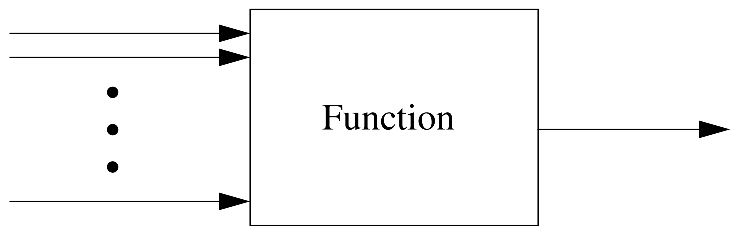
int (total_HK_dollars * DOLLAR2CENTS)

How to Run a Program?



Functions

- o facilitate top-down design
  - decomposition
  - information hiding
  - easily understandable
  - easily modifiable
- o reusability



Functions (Cont'd)

- Function Prototype (**.h** files)

```
<type> <function identifier>(<type list>);
```

```
<type>, <type>, •••
```

- Function Definition(**.cpp** files)

```
<type> <function identifier>(<parameters>)
{
  <local declarations>
  <sequences of statements>
}
```

```
<type> <identifier>, •••
```

- Call

```
<function name>(<arguments>)
```

  - o result of function call is a value of type <type>
  - o arguments match parameters

Some Mathematical Functions in math.h

	Purpose
• log(x)	ln(x) - natural log of x (base e) for x>0.
• log10(x)	base 10 log of x for x>0.
• exp(x)	e to the power x, where e=2.71823
• pow(x,y)	x to the power y
• sqrt(x)	positive square root of x
• ceil(x)	smallest integer not less than x
• floor(x)	largest integer not greater than x
• sin(x), cos(x), tan(x),•••	

Example

- Example program:

```
// Convert inches to feet and inches
// Input: inches
// Output: feet and inches

#include <iostream.h>

void main() {

     /* inches to feet conversion factor */
     const int in2feet = 12;

     int  inches; /* number of inches */

     cout<< "Enter number in inches: ";
     cin >> inches;

     cout<< inches / in2feet << " feet "
         << inches % in2feet << " inches "
         << endl;
}
```

Example (Cont'd)

```
// File main.cpp
// Input  inches
// Output feet and inches

#include <iostream.h>

// Function Prototype
int feet(int);
int rinches(int);


void main() {
     int  inches;       // Number of inches

     cout << "Inches: ";
     cin >> inches;
     cout << feet(inches) << " feet  "
          << rinches(inches) << " inches"
          << endl;
}

// Functions for Converting inches to feet
const int i2ffactor = 12;  // conversion factor
int feet(int inches) {
     return inches / i2ffactor;
}
int rinches(int inches) {
     return inches % i2ffactor;
}
```

Example (Cont'd)

```
// File i2f.h
int feet(int);
int rinches(int);


// File i2f.cpp
// Functions for Converting inches to feet
const int i2ffactor = 12;  // conversion factor
int feet(int inches) {
     return inches / i2ffactor;
}
int rinches(int inches) {
     return inches % i2ffactor;
}


// File main.cpp
// Input  inches
// Output feet and inches

#include <iostream.h>
#include "i2f.h"

void main() {
     int  inches;       // Number of inches

     cout << "Inches: ";
     cin >> inches;
     cout << feet(inches) << " feet   "
          << rinches(inches) << " inches"
          << endl;
}
```

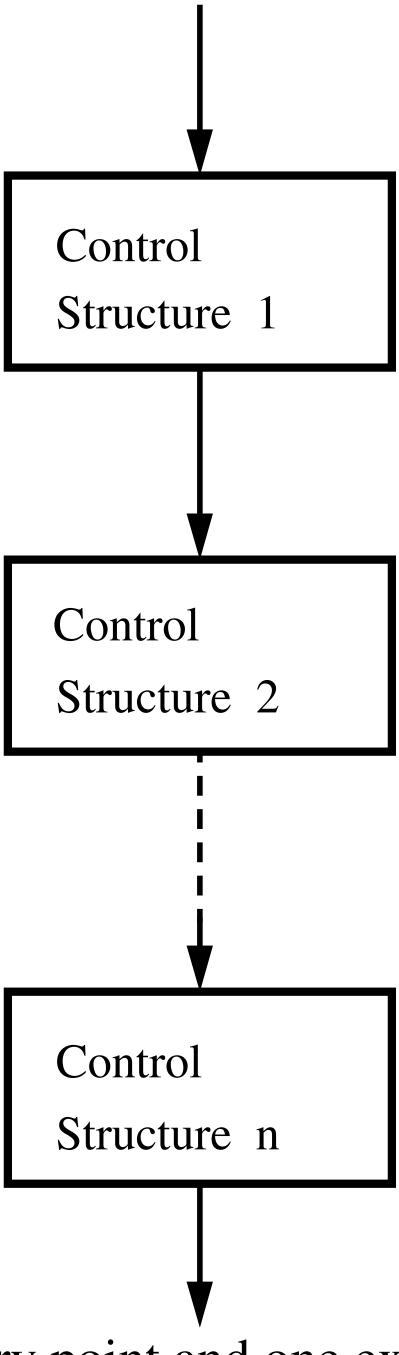
Structured Programming

- disciplined approach to programming
- easy to read and understand
- less likely to contain errors

Structured programming utilizes **control structures** to control the flow of statement execution in a program.

- sequence
- selection
- iteration

Write a program as a sequence of control structures rather than as a sequence of individual statements.



one entry point and one exit point.

The three basic sequences of processing

1) Sequence

For example:

```
Add 1 to page_count
Print heading line
Set linecount to zero
Read customer record
```

Generic example:

```
do T ask P
do T ask Q
do T ask R
```

2) Selection

Generic example:

```
if condition is true then
   perform P
otherwise
   perform Q
end

if condition then
   perform R
end
```

3) Repetition

```
while condition is true
   do T ask P
   do T ask Q
   do T ask R
end
```

Relational Expression

Examples:

NumberOfStudents < 200

10 > 20

20 * J == 10 + I

Relational Operators:

<u>Math</u>	<u>C++</u>
$=$	==
$<$	<
$\leq$	<=
$>$	>
$\geq$	>=
$\neq$	!=

Operator precedence:

`*   /   %`

`+   -`

`<   <=   >=   >`

`==   !=`

Logical Operator

To form more complex logical expressions or conditions.

Example:

(NumberOfStudent > 100) && (NumberOf Student < 200)

! (NumberOfStudent < 100)

(I >= 10) || (I <= 0)

Truth table for "&&":

Operand1	Operand2	Operand1 && Operand2
1 (True)	1 (True)	1 (True)
1 (True)	0 (False)	0 (False)
0 (False)	1 (True)	0 (False)
0 (False)	0 (False)	0 (False)

Truth table for "or" and "not":

•••

Logical Equivalence

Example:

! ((I >= 10) || (I <= 0))

is the same as

? and ?

! ((I > 10) && (I <= 20))

is the same as

? or ?

Value of logical expression

! (X || Y)	is the same as	(! X) && (! Y)
! (X && Y)	is the same as	(! X) || (! Y)
! (! X)	is the same as	X

Operator Precedence

!

*   /   %

\+   -

<   <=   >=   >

==   !=

&&

||

=

Examples:

(I > 0) && (I <= 10) || (I > 20) && (I <= 30)

is the same as

((I > 0) && (I <= 10)) || ((I > 20) && (I <= 30))

Suppose X, Min and Max are are of type float,

Min <= X && X <= Max

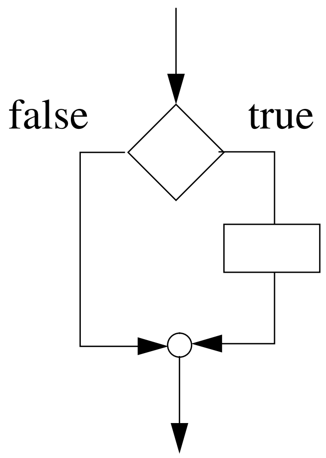
is the same as

?

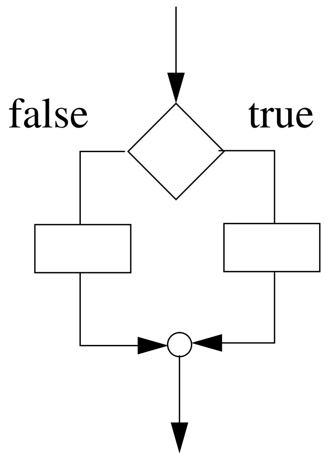
Again, *use parentheses* to clarify.

If Statement

```
if (<condition>) {
  <sequence of statements>
}
```




```
if (<condition>) {
  <sequence of statements>
} else {
  <sequence of statements>
}
```



If Statement (Cont'd)

Example:

```
// Compare two input integers

#include <iostream.h>

main() {
     int First,      // input first number
         Second;     // input second number

     cout << "Input two numbers: ";
     cin >> First >> Second;

     if (First > Second)
          cout << First
               << " is greater than "
               << Second << endl;
     else if (First < Second)
          cout << First
               << " is less than "
               << Second << endl;
     else
          cout << First
               << " is equal to "
               << Second << endl;

     return 0;
}
```