MA144: Problem Solving and Computer Programming

Lecture-7

Tokens, Operators

Character Set

32		56	8	80	P	104	h
33	!	57	9	81	Q	105	i
34	"	58	:	82	R	106	j
35	#	59	;	83	S	107	k
36	$	60	<	84	T	108	l
37	%	61	=	85	U	109	m
38	&	62	>	86	V	110	n
39	'	63	?	87	W	111	o
40	(	64	@	88	X	112	p
41	)	65	A	89	Y	113	q
42	*	66	B	90	Z	114	r
43	+	67	C	91	[	115	s
44	,	68	D	92	\	116	t
45	–	69	E	93	]	117	u
46	.	70	F	94	^	118	v
47	/	71	G	95	_	119	w
48	0	72	H	96	'	120	x
49	1	73	I	97	a	121	y
50	2	74	J	98	b	122	z
51	3	75	K	99	c	123	{
52	4	76	L	100	d	124	|
53	5	77	M	101	e	125	}
54	6	78	N	102	f	126	~
55	7	79	O	103	g		

Tokens

Tokens - the smallest individual units in a program

- Keywords
- Identifiers
- Constants
- Strings
- Operators
- Special symbols

A C++ program is written using these tokens, white spaces, and the syntax of the language.

The syntax for a programming language is the set of grammar rules for that language.

Keywords

- Reserved identifiers (names) and cannot be used as names for the program variables or other user-defined program elements
- Because they have predefined meaning in C++

```
alignas
alignof
asm
auto
bool
break
case
catch
char
class
const
const_cast
constexpr
continue
decltype
default
delete
do
double
dynamic_cast
else
enum
explicit
export
extern
false
float
for
friend
goto
if
inline
int
log
long
mutable
namespace
new
noexcept
nullptr
operator
private
protected
public
register
reinterpret_cast
return
short
signed
sizeof
static
static_assert
static_cast
struct
switch
template
this
thread_local
throw
true
try
typedef
typeid
typename
union
unsigned
using
virtual
void
volatile
wchar_t
while
```

Keywords (contd…)

- words like cin and cout are not on the list of keywords
- you are allowed to redefine these words,
- hence they are not keywords
- Avoid re-define these words

Identifiers

- Identifiers refer to the names of variables, functions, arrays, classes, etc., created by the programmer.
- They are the fundamental requirement of any language.
- Each language has its own rules for naming these identifiers.

Rules to follow…

- Only alphabetic characters, digits and underscores are permitted.
- The name cannot start with a digit (but can start with underscore).
- Uppercase and lowercase letters are distinct.
- A declared keyword cannot be used as a variable name.

x
x1
x_1 _abc
ABC123z7
sum
RATE
count
data2
Big_Bonus

12
3X
%change
data-1
myfirst.c
PROG.CPP
Data abc

use meaningful identifiers as name of the identifier

Variable Declarations

First rule of variable use:

Must declare a variable (by specifying its type and name) before using it anywhere in your program

- All variable declarations should ideally be at the beginning of the main() or other functions

ends with semicolon

Syntax

```
type var_name_1, var_name_2, var_name_3;
```

type – what kind of data we will be storing in the variable

Examples

```
int sum, a, b;
float average, x, y, z;
```

- compiler implements variables as memory locations
- the value of a variable is stored in the memory location assigned to that variable
- memory is a list of consecutive storage locations, each having a unique address
- a variable is mapped to a location of the memory, called its address
- the value of a variable can be changed during the execution of the program
- A variable can have only one value assigned to it at any given time during the execution of the program

Assignment Statements

Assignment - assigning a value to a variable

Syntax `variable = expression;`

The **assignment statement** instructs the computer that set the value of the expression on RHS to the variable on LHS

i.e., store the value of the expression in the memory location, which is named `variable`

Examples

```
sum=a+b;
count=cout+1;
  a=4;
```

= is an assignment operator

Initializing a variable

We can initialize a variable at the time of variable declaration

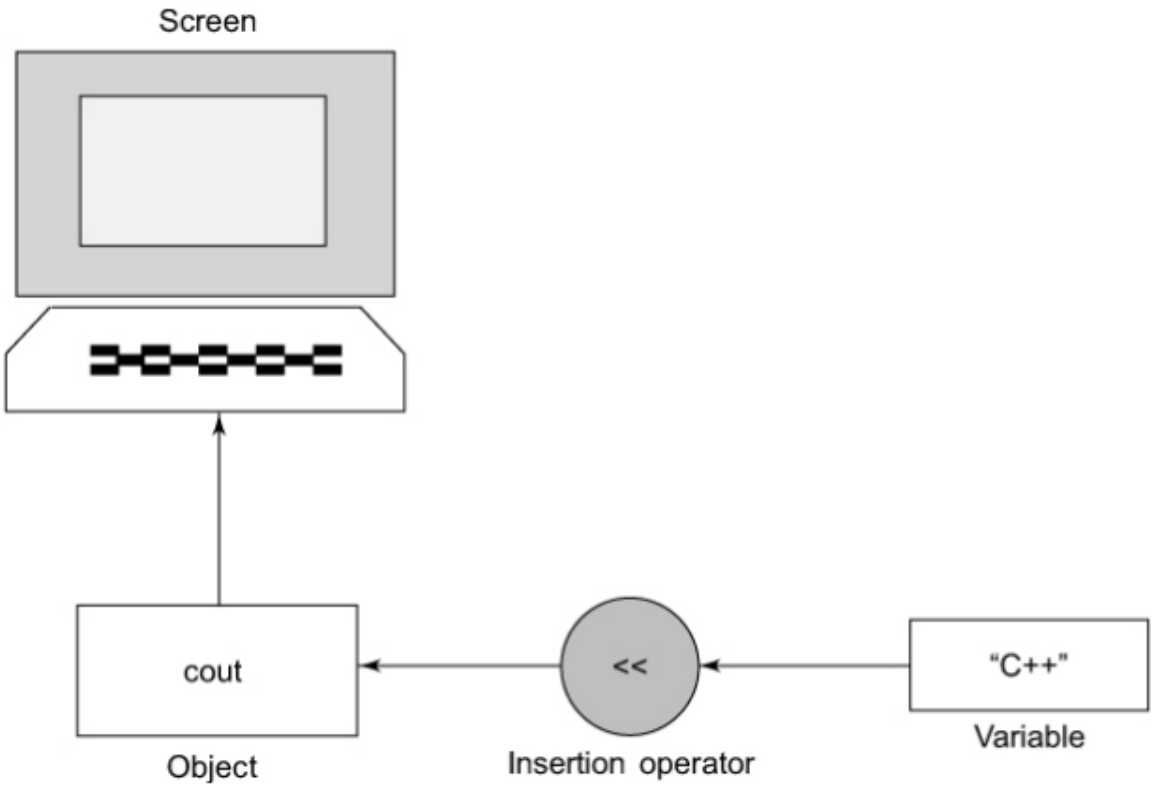
```
int count=5;

int sum=a+b;
```

Output using cout object

- The operator << is called the insertion or put to operator.
- It inserts (or sends) the contents of the variable on its right to the object on its left

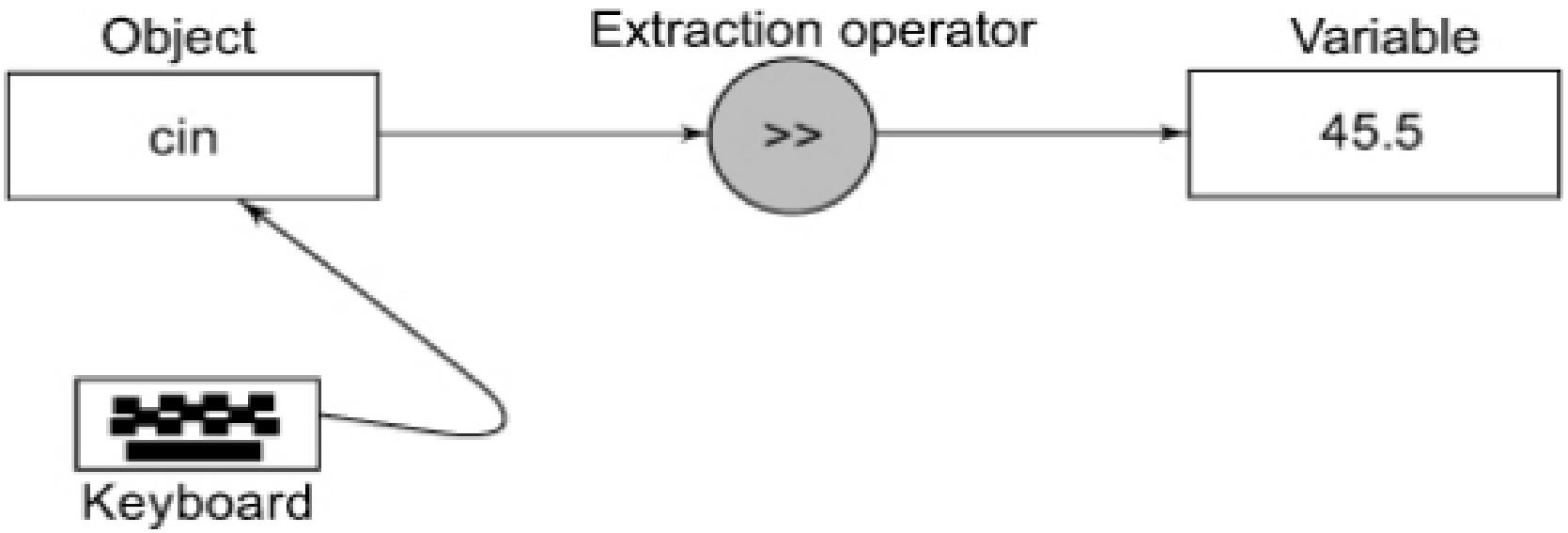
```
cout << “Hellow world”;
  cout << sum;
```



Input using `cin` object

- The operator >> is known as extraction or get from operator.
- It extracts (or takes) the value from the keyboard and assigns it to the variable on its right

```
cin  >> sum;
```




```
cin  >> var1 >> var2 >> var3;
```

Designing Input and Output

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, sum;
    cout << "Enter two numbers: ";
    cin >> a >> b;
    sum=a+b;
    cout<<"the sum of "<< a<<" and "<<b<<" is equal to "<<sum;
    return 0;
}
```

Enter two numbers: 10 20

the sum of 10 and 20 is equal to 30

Suggestion

In the laboratory

- As far as possible, do not develop programs for single set of input data

  After generating the output for some input data, check whether the program should be executed again with another set of data.

VERY IMPORTANT – Use proper indentation for better readability of the programs.

MORE IMPORANT – Document every program and every important statement by way of comments

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  { int a=4, b=5, sum;
5      cout<< a<<'\n'<< endl;  /* these are comments*/
6      cout<<100<<'\n'<<"100";  // these are comments
7      return 0;
8  }
9
```

Escape Sequences

- Newline **\n**
- Horizontal tab **\t**
- Backslash ****
- Double quote **\”**

Starting newlines in output

```
#include <iostream>
using namespace std;
int main()
{
    cout<<"hai \n";
    cout<<"hai"<<"\n";
    cout<<"hai"<<'\n';
    cout<<"hai"<<endl;
    cout<<100;
    return 0;
}
```

Output:

hai
hai
hai
hai
100

Data Types

- **Integer type**
- **Floating-point type**

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      double a,b;
6      cout<<"enter a number: ";
7      cin>> a;
8      cout<< a<<endl;
9      cout<<"enter a number: ";
10     cin>> b;
11     cout<< b;
12     return 0;
13 }
```

2.34568×10^{11}

3.45679×10^{-09}

```
enter a number: 234567898989
2.34568e+11
enter a number: 0.000000034567898989
3.45679e-09
```

The exponent after the **e** definitely must **not** contain a decimal point.

Some Number Types

Type Name	Memory Used	Size Range
short (also called *short int*)	2 bytes	-32,768 to 32,767
int	4 bytes	-2,147,483,648 to 2,147,483,647
long (also called *long int*)	4 bytes	-2,147,483,648 to 2,147,483,647
float	4 bytes	approximately 10^{-38} to 10^{38}
double	8 bytes	approximately 10^{-308} to 10^{308}
long double	10 bytes	approximately 10^{-4932} to 10^{4932}

Arithmetic Operators

Binary operators : + - * / %

- The % operator can not be applied to floating-point type

```
#include<iostream>
using namespace std;

int main()
{
cout<<4/3<<'\t'<<4/3.0<<"\t"<<4.0/3<<'\t'<<4.0/3.0<<endl;

cout<<14%3<<endl;

cout<<4*3<<'\t'<<4*3.2<<"\t"<<4.2*3<<'\t'<<4.2*3.0<<endl;

cout<<4+3<<'\t'<<4+3.2<<"\t"<<4.2+3<<'\t'<<4.2+3.0<<endl;

return 0;
}
```

```cpp
#include<iostream>
using namespace std;

int main()
{
cout<<4/3<<'\t'<<4/3.0<<"\t"<<4.0/3<<'\t'<<4.0/3.0<<endl;

cout<<14%3<<endl;

cout<<4*3<<'\t'<<4*3.2<<"\t"<<4.2*3<<'\t'<<4.2*3.0<<endl;

cout<<4+3<<'\t'<<4+3.2<<"\t"<<4.2+3<<'\t'<<4.2+3.0<<endl;

return 0;
}
```

```
1       1.33333 1.33333 1.33333
2
12      12.8    12.6    12.6
7       7.2     7.2     7.2


--------------------------------
```

- C++ allows only one kind of parentheses ( ) in arithmetic expressions.
- The other varieties are reserved for other purposes.
- If you omit parentheses, the computer will follow some rules called **precedence rules** to evaluate an expression.

* multiplication / division % remainder (modulo)	**Highest precedence**
+ addition - subtraction	**Lowest precedence**

Observe: $x^3 = x * x * x$

- Arithmetic operators are executed left to right when operators have the same precedence

```
#include<iostream>
using namespace std;

int main()
{
cout<<2+4*5/2*8-3<<endl;

cout<<10-3+16%3*5+4-8/5*10;

return 0;
}
```

What is the output of the above program?

Combining Assignment Operator with Arithmetic Operators

```
Variable Op= Expression
```

is equivalent to

```
Variable = Variable Op (Expression)
```

Example	Equivalent to:
count += 2;	count = count + 2;
total –= discount;	total = total – discount;
bonus *= 2;	bonus = bonus * 2;
time /= rush_factor;	time = time / rush_factor;
change %= 100;	change = change % 100;
amount *= cnt1 + cnt2;	amount = amount * (cnt1 + cnt2);

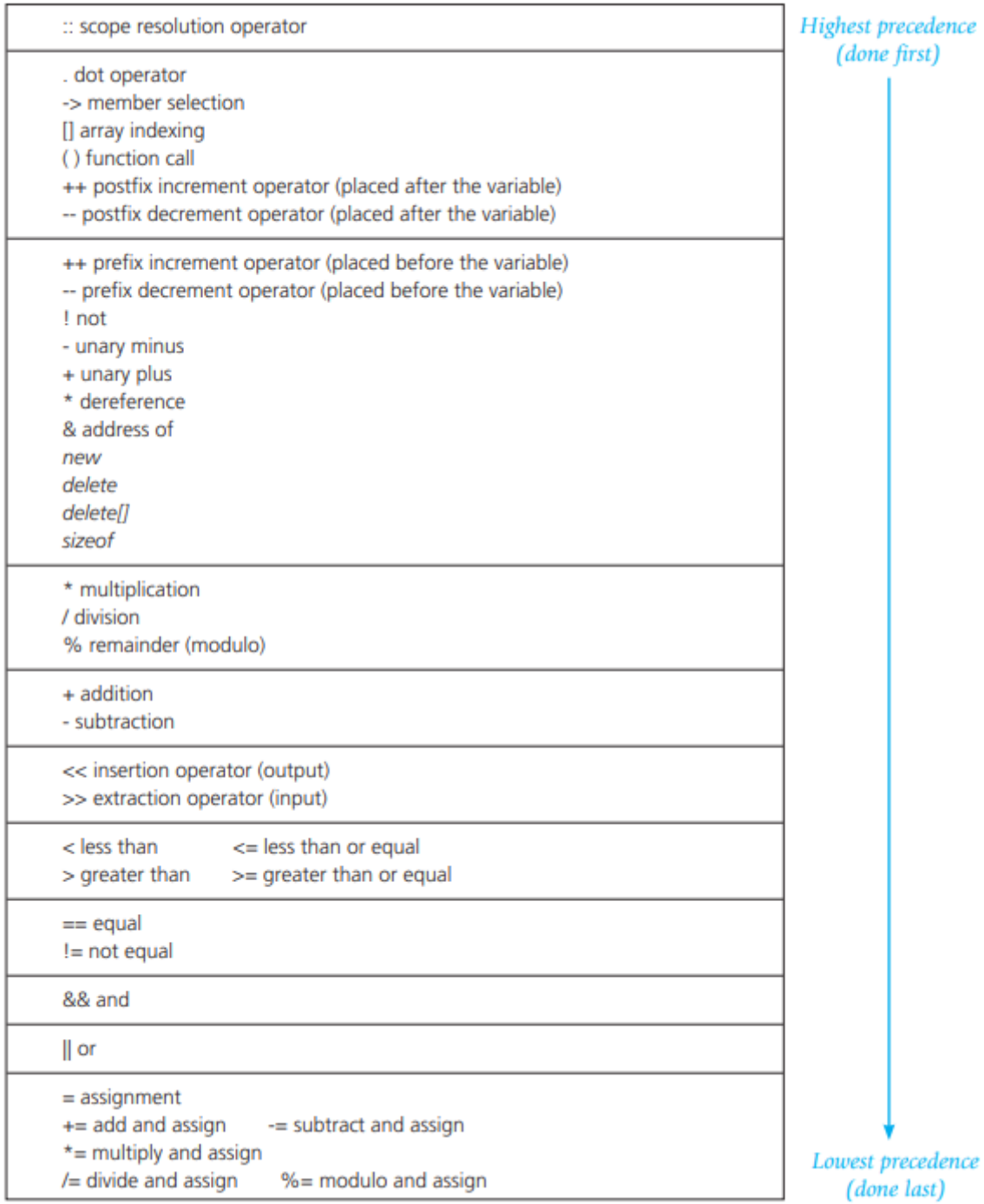
Comparison Operators

Math Symbol	English	C++ Notation
=	equal to	==
≠	not equal to	!=
<	less than	<
≤	less than or equal to	<=
>	greater than	>
≥	greater than or equal to	>=

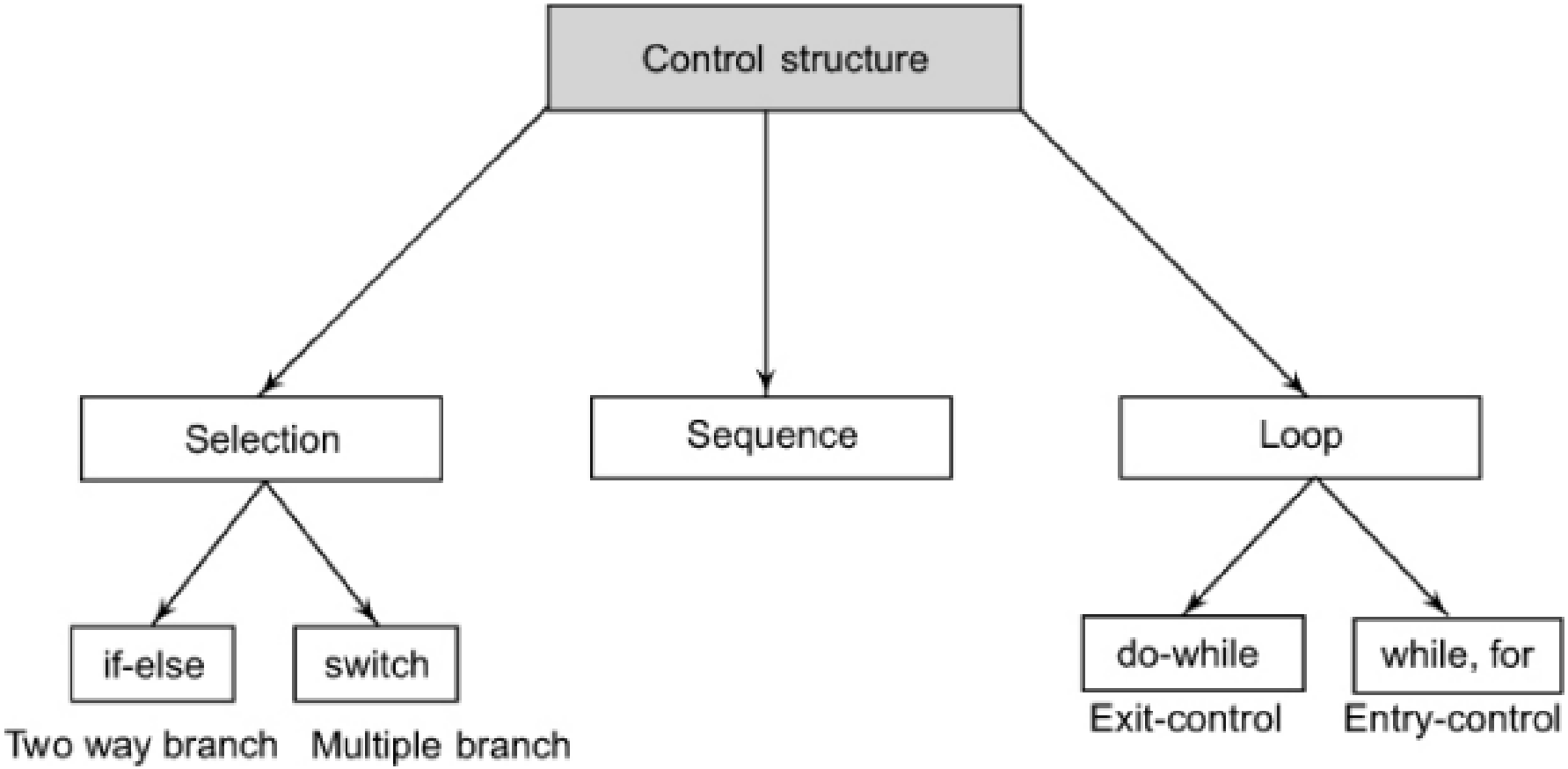
< <= > >= Highest precedence

Left-right associative

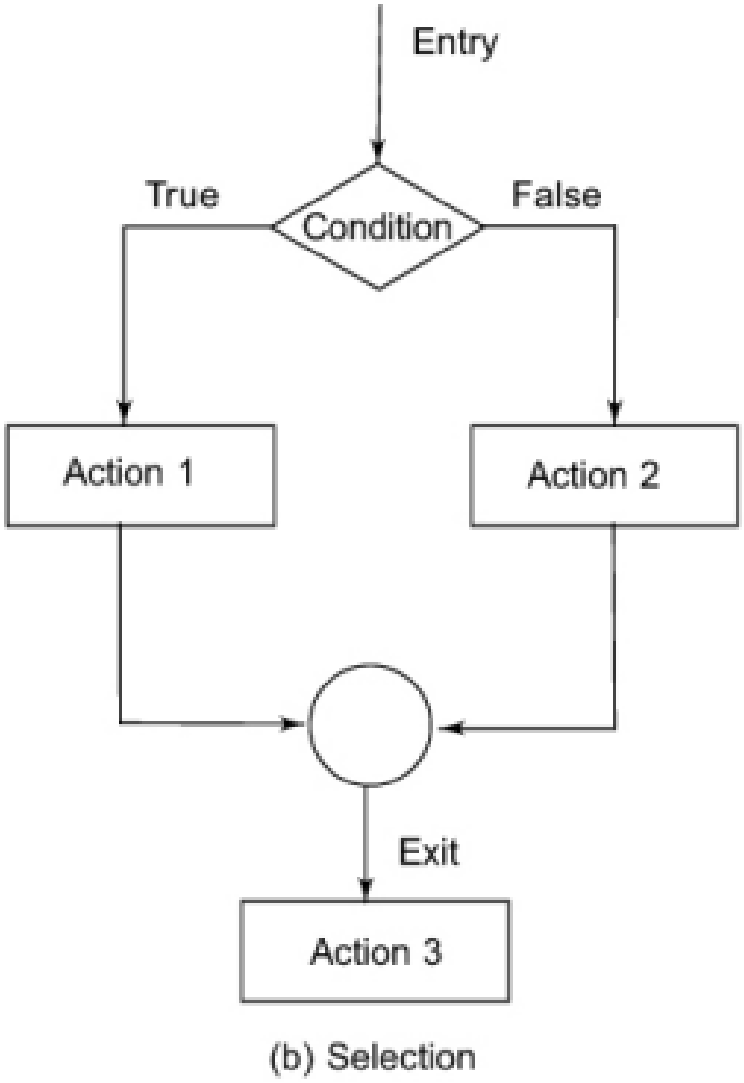
== != Lowest precedence


:: scope resolution operator
. dot operator
-> member selection
[] array indexing
( ) function call
++ postfix increment operator (placed after the variable)
-- postfix decrement operator (placed after the variable)
++ prefix increment operator (placed before the variable)
-- prefix decrement operator (placed before the variable)
! not
- unary minus
+ unary plus
* dereference
& address of
new
delete
delete[]
sizeof
* multiplication
/ division
% remainder (modulo)
+ addition
- subtraction
<< insertion operator (output)
>> extraction operator (input)
< less than
<= less than or equal
> greater than
>= greater than or equal
== equal
!= not equal
&& and
|| or
= assignment
+= add and assign
-= subtract and assign
*= multiply and assign
/= divide and assign
%= modulo and assign
Highest precedence
(done first)
Lowest precedence
(done last)

Control Structures



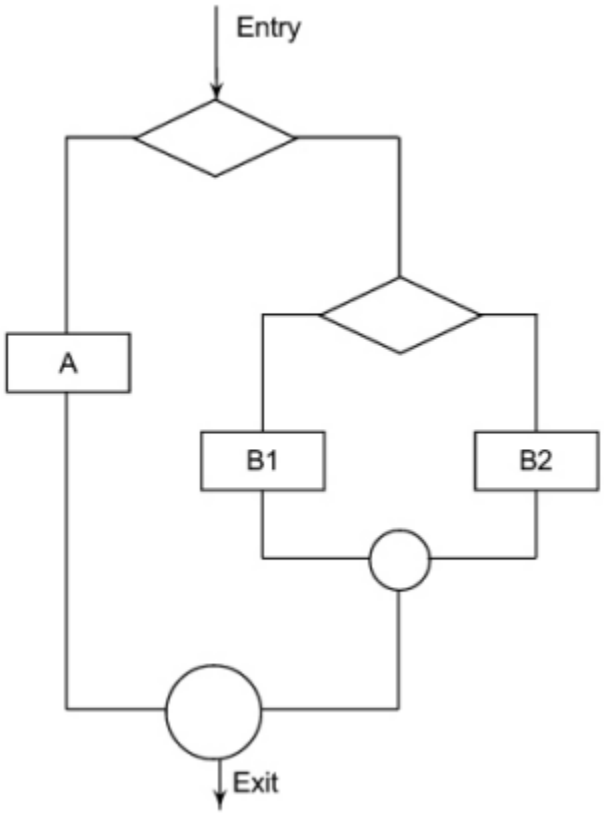
if-else

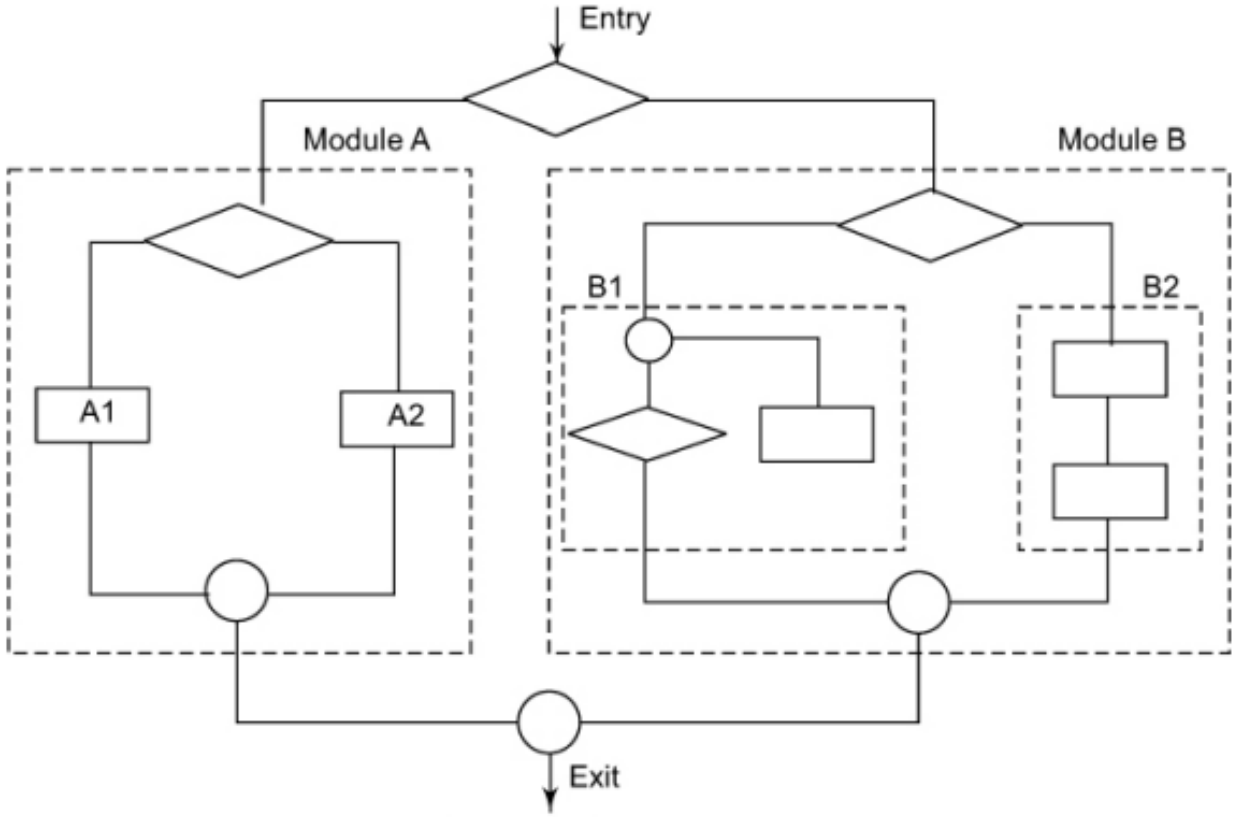


(b) Selection

```
if (Boolean expression)
 {
        statement1;
        statement2;
   }
else
 {
        statement3;
        statement4;
   }
Statement5;
Statement6;
```

A Boolean expression is any expression that is either true or false.


Entry
A
B1
B2
Exit


Entry
Module A
Module B
A1
A2
B1
B2
Exit