Programming Fundamentals

LECTURE 3

Cin (standard input)

C++ uses cin stream object to get standard input.

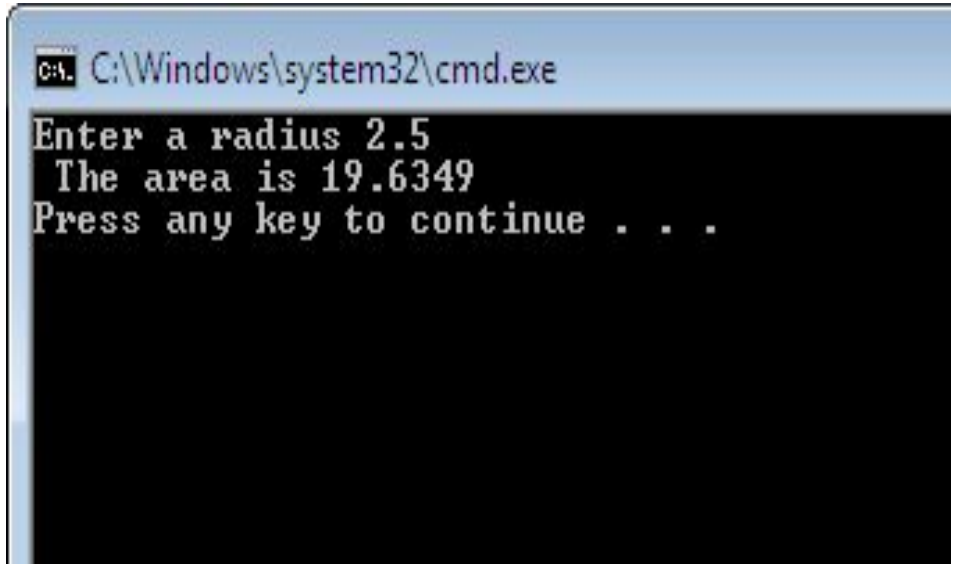
The word cin stands for console input

Syntax

- cin >> variable
- >> is known as extraction operator or get from operator. It gets the input from cin object
- cin object then store the input in the variable
- cin >> a >> b >> c;
- The above line will get three values from keyboard and store it in variables a, b and c

Reading input from keyboard

```
#include <iostream>

using namespace std;

int main()

{

double radius;

cout << “Enter a radius ”;

cin >> radius;

double area=radius*radius*3.14159;

cout << “ The area is “ << area << endl;

return 0;

}
```



Numeric operators

SYMBOL	OPERATION	EXAMPLE	VALUE
+	addition	`7 + 3;`	10
–	subtraction	`7 – 3;`	4
*	multiplication	`7 * 3;`	21
/	division	`7 / 3;`	2
%	modulus	`7 % 3;`	1

Shorthand assignment operator

Long Hand	Short Hand
x = x * y;	x *= y;
x = x / y;	x /= y;
x = x % y;	x %= y;
x = x + y;	x += y;
x = x - y;	x -= y;

Operator precedence

The order in which different types of operators in an expression are evaluated is known as operator precedence

Precedence rules determine the order in which operations are performed:

- Expressions grouped in parentheses are evaluated first.
- unary −
- *, /, and %
- + and −

Operators within the same level are evaluated in left-to-right order (w/o parentheses).

Example

These are the same rules used in mathematical expressions, so they should feel natural.

When in doubt, add parentheses for clarity!

```
24 / 7 + 5      --->   3 + 5      --->   8

24 / (7 + 5)    --->  24 / 12     --->   2

1 + 2 - 3 - 4   --->  3 - 3 - 4   --->  0 - 4   ---> -4

4 * 3 - 2       --->  12 - 2      --->  10
```

Increment & decrement operators

++var

- Preincrement
- Increment var by 1 and use the new var value
- Example:
  - Assume i = 1
  - Int j= ++i ; // j is 2 i is 2

var++

- Postincrement
- Increment var by 1 but use the original var value
- Example:
  - Assume i = 1
  - Int j= i++ ; // j is 1 i is 2

Increment & decrement operators

--var

- Predecrement
- Decrement the var by 1 and use the new var value
- Example:
  - Assume i = 1
  - Int j= --i ; // j is 0 i is 0

var--

- Postdecrement
- Decrement the var by 1 and use the original var value
- Example:
  - Assume i = 1
  - Int j= i-- ; // j is 1 i is 0

Example

```
int x = 7, y = 3;
cout << x << " " << y << endl;
cout << ++x << " " << --y << endl;
cout << x << " " << y << endl;
cout << x++ << " " << y-- << endl;
cout << x << " " << y << endl;
```

Numeric type conversion

Typecasting is the concept of converting the value of one type into another type. For example, you might have a float that you need to use in a function that requires an integer.

Two types of conversions

- Implicit
- Explicit

Implicit type casting

When the type of the source is NOT the same as the type of the target variable, errors may result

When a decimal value is assigned to an integer

variable, the decimal value is automatically

truncated to an integer value when it is stored:

```
const int NUMTESTS = 2;
 double Test1 = 93.0,
        Test2 = 86.0;
 int    testAverage;
 testAverage = (Test1 + Test2)/NUMTESTS; // testAverage <--- 89
                                         //     not 89.5
```

Implicit type casting

When an integer value is assigned to a decimal variable, the integer value is automatically " widened" to a decimal value when it is stored:

```
double Sum;
int X = 17, Y = 25, Z = 42;
Sum = X + Y + Z; // Sum <--- 84.0, not 84
```

Explicit type casting

Sometimes a programmer need to convert a value from one type to another type in a situation where the compiler will not do it automatically

There are several kinds of casts in standard c++

- Static cast

Explicit casting

```
static_cast<dataTypeName>(expression)
```

```
static_cast<int>(7.9)          7
static_cast<int>(3.3)          3
static_cast<double>(25)        25.0
static_cast<double>(5+3)       =static_cast<double>(8)=8.0
static_cast<double>(15)/2      =15.0/2
                               (because static_cast<double>(15)=15.0)
                               =15.0/2.0=7.5
static_cast<double>(15/2)      =static_cast<double>(7) (because 15/2=7)
                               = 7.0
static_cast<int>(7.8 +
static_cast<double>(15)/2)     =static_cast<int>(7.8+7.5)
                               =static_cast<int>(15.3)
                               = 15
static_cast<int>(7.8 +
static_cast<double>(15/2))     =static_cast<int>(7.8 + 7.0)
                               =static_cast<int>(14.8)
                               = 14
```

Example

```
#include <iostream>

using namespace std;

int main()

{

int nValue1 = 10;

int nValue2 = 4;

float fValue = static_cast<float>(nValue1) / nValue2;

cout<< fValue;

return 0;

}
```

Selection

Control structure

A computer can proceed:

- In sequence
- Selectively (branch) - making a choice
- Repetitively (iteratively) - looping

Some statements are executed only if certain conditions are met

A condition is met if it evaluates to `true`

flowchart

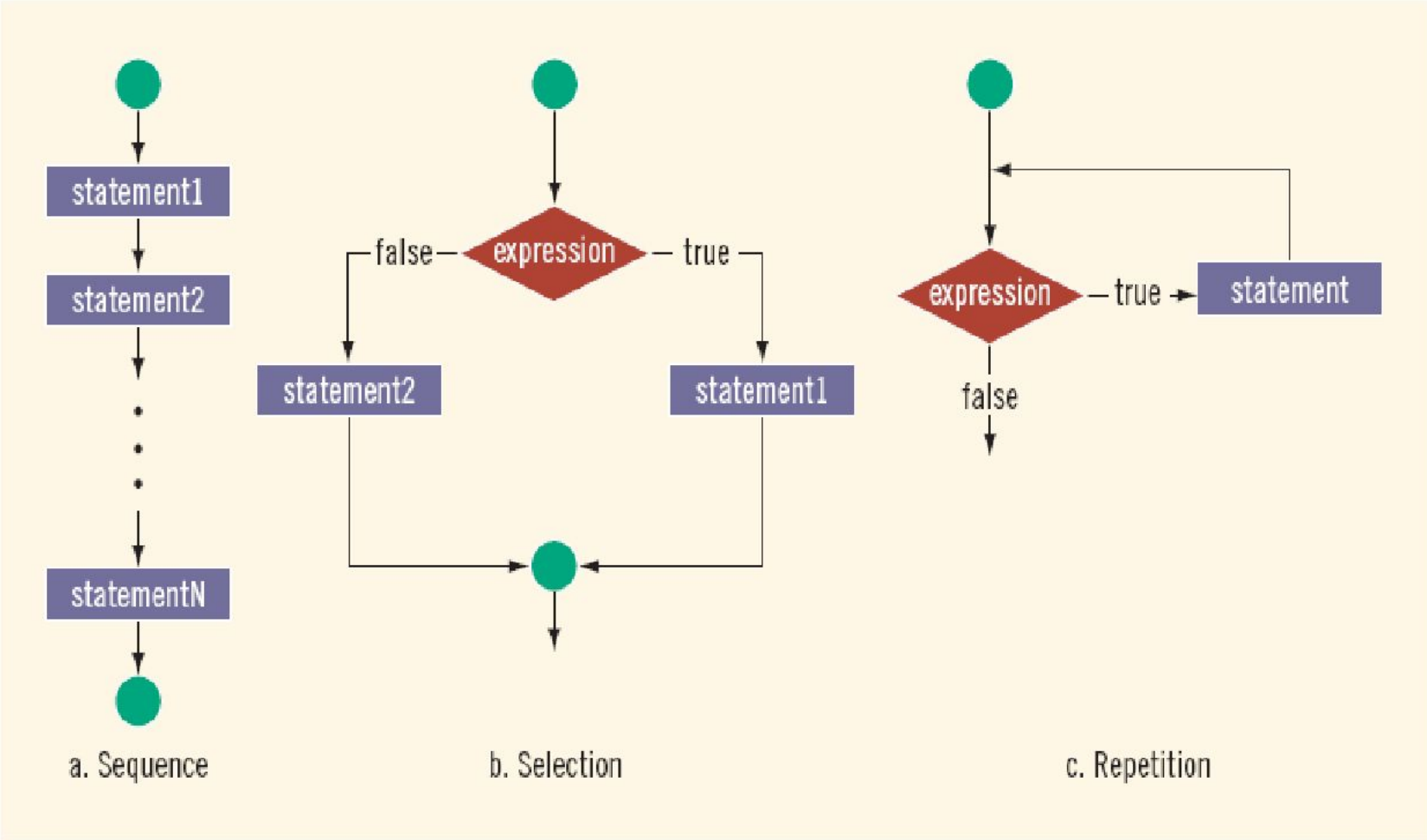


FIGURE 4-1 Flow of execution

Relational operators

A condition is represented by a logical (Boolean) expression that can be `true` or `false`

Relational operators:

- Allow comparisons
- Require two operands (binary)
- Evaluate to `true` or `false`

Relational operator

TABLE 4-1 Relational Operators in C++

Operator	Description
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Relational Operators and Simple Data Types

You can use the relational operators with all three simple data types:

- `8 < 15` evaluates to `true`
- `6 != 6` evaluates to `false`
- `2.5 > 5.8` evaluates to `false`
- `5.9 <= 7.5` evaluates to `true`

Comparing Characters

TABLE 4-2 Evaluating Expressions Using Relational Operators and the ASCII Collating Sequence

Expression	Value of Expression	Explanation
`' ' < 'a'`	`true`	The ASCII value of `' '` is 32, and the ASCII value of `'a'` is 97. Because 32 < 97 is `true`, it follows that `' ' < 'a'` is `true`.
`'R' > 'T'`	`false`	The ASCII value of `'R'` is 82, and the ASCII value of `'T'` is 84. Because 82 > 84 is `false`, it follows that `'R' > 'T'` is `false`.
`'+' < '*'`	`false`	The ASCII value of `'+'` is 43, and the ASCII value of `'*'` is 42. Because 43 < 42 is `false`, it follows that `'+' < '*'` is `false`.
`'6' <= '>'`	`true`	The ASCII value of `'6'` is 54, and the ASCII value of `'>'` is 62. Because 54 <= 62 is `true`, it follows that `'6' <= '>'` is `true`.

Logical operators

TABLE 4-5 Logical (Boolean) Operators in C++

Operator		Description
`!`	← unary	`not`
`&&`	← binary	`and`
`||`	← binary	`or`

Expression	Value	Explanation
`!('A' > 'B')`	`true`	Because `'A' > 'B'` is `false`, `!('A' > 'B')` is `true`.
`!(6 <= 7)`	`false`	Because `6 <= 7` is `true`, `!(6 <= 7)` is `false`.

Expression	Value	Explanation
`(14 >= 5) && ('A' < 'B')`	`true`	Because `(14 >= 5)` is `true`, `('A' < 'B')` is `true`, and `true && true` is `true`, the expression evaluates to `true`.
`(24 >= 35) && ('A' < 'B')`	`false`	Because `(24 >= 35)` is `false`, `('A' < 'B')` is `true`, and `false && true` is `false`, the expression evaluates to `false`.

Expression	Value	Explanation
`(14>=5) || ('A'>'B')`	`true`	Because `(14 >= 5)` is `true`, `('A' > 'B')` is `false`, and `true || false` is `true`, the expression evaluates to `true`.
`(24>=35) || ('A'>'B')`	`false`	Because `(24 >= 35)` is `false`, `('A' > 'B')` is `false`, and `false || false` is `false`, the expression evaluates to `false`.
`('A'<='a') || (7 !=7)`	`true`	Because `('A' <= 'a')` is `true`, `(7 != 7)` is `false`, and `true || false` is `true`, the expression evaluates to `true`.

Precedence of operators

Operators	Precedence
!, +, – (unary operators)	first
*, /, %	second
+, –	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

One-Way Selection

The syntax of one-way selection is:

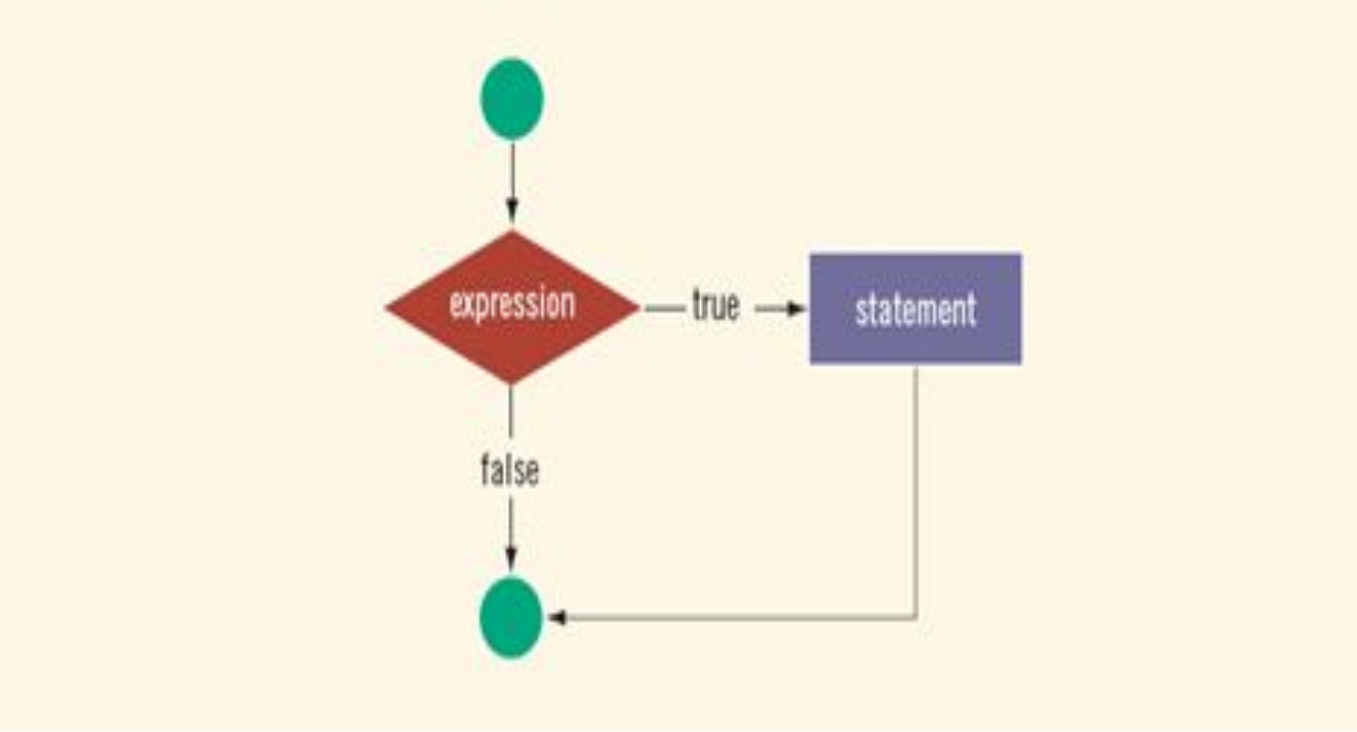
```
if (expression)
    statement
```

The statement is executed if the value of the expression is `true`

The statement is bypassed if the value is `false`; program goes to the next statement

`if` is a reserved word

If statement



explanation

```
if (score >= 60)
    grade = 'P';
```

In this code, if the expression (`score >= 60`) evaluates to `true`, the assignment statement, `grade = 'P';`, executes. If the expression evaluates to `false`, the statements (if any) following the `if` structure execute. For example, if the value of `score` is 65, the value assigned to the variable `grade` is `'P'`.

Two-Way Selection

Two-way selection takes the form:

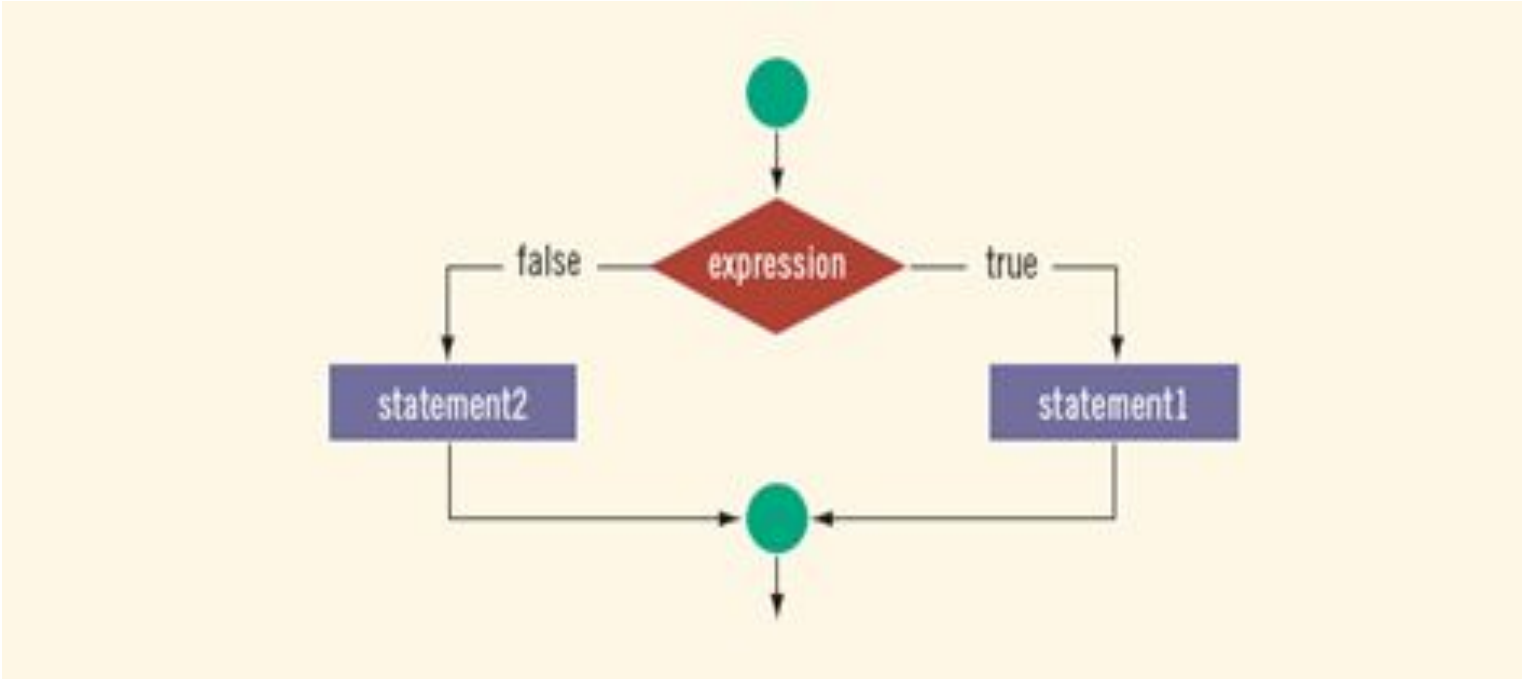
```
if (expression)
    statement1
else
    statement2
```

If expression is `true`, `statement1` is executed; otherwise, `statement2` is executed

- `statement1` and `statement2` are any C++ statements

`else` is a reserved word

Two-way selection



Compound statement

```
if (age > 18)
{
 cout << "Eligible to vote." << endl;
 cout << "No longer a minor." << endl;
}
else
{
 cout << "Not eligible to vote.“ << endl;
 cout << "Still a minor." << endl;
}
```

Sample program

```
#include <iostream>
using namespace std;
int main(){
int number;
cout<<"Enter number\n";
cin>>number;
if(number%2==0)
        cout<<number<<" is even\n";
else
        cout<<number<<" is odd\n";
return 0; }
```

If else if

```
if(con1)
{
	code1
}
else if(con2)
{
	code2
}
else if(con3)
{
	code3
}
else
{
	coden
}
```

```
#include <iostream>
using namespace std;
int main()
{
char grade;
float score;
cin >> score;
if(score>=90.0)
grade='A';
else if(score>=80.0)
grade='B';
else if(score>=70.0)
grade='C';
else if (score>=60.0)
grade='D';
else
grade='F';
cout<<grade<<endl;
return 0;
}
```

Conditional Operator (?:)

Conditional operator (`?:`) takes three arguments

- Ternary operator

Syntax for using the conditional operator:

```
expression1 ? expression2 :
expression3
```

If `expression1` is `true`, the result of the conditional expression is `expression2`

- Otherwise, the result is `expression3`

Conditional Operator (?:)

```
if(a>=b)
 Max=a;
else
 Max=b;
**********************
Max=(a>=b)?a:b;
```

switch Structures

switch structure: alternate to if-else

switch (integral) expression is evaluated first

Value of the expression determines which corresponding action is taken

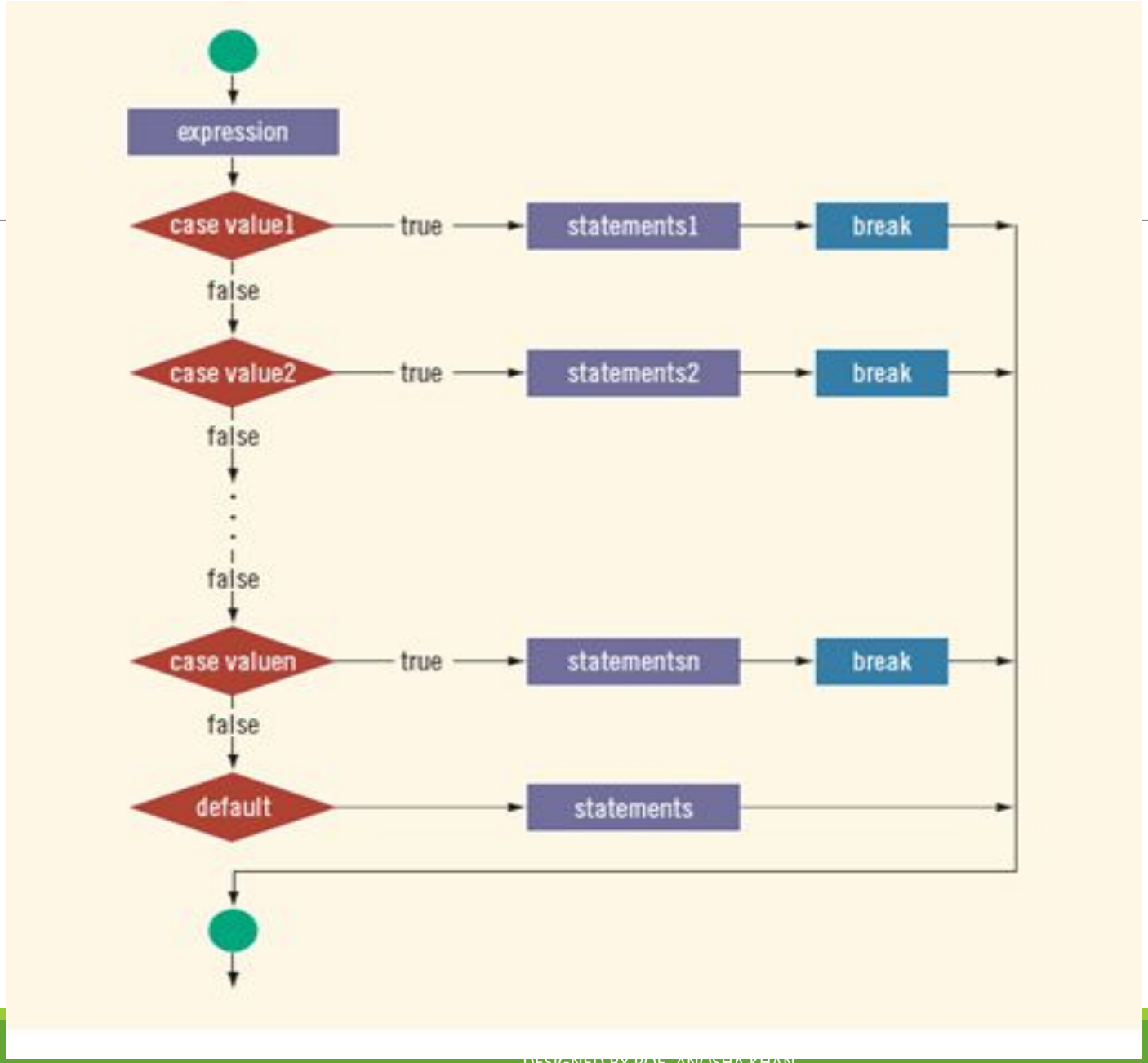
Expression is sometimes called the selector

```
switch (expression)
{
case value1:
    statements1
    break;
case value2:
    statements2
    break;
    .
    .
    .
case valuen:
    statementsn
    break;
default:
    statements
}
```

explanation

switch works as follows:-

- The expression, just test in this case, is evaluated.
- The case labels are checked in turn for the one that matches the value.
- If none matches, and the optional default label exists, it is selected,
- the break statement is normally added before the next case label to transfer control out of the switch statement.


expression
case value1
true
statements1
break
false
case value2
true
statements2
break
false
false
case valuen
true
statementsn
break
false
default
statements

`switch` Structures (continued)

One or more statements may follow a case label

Braces are not needed to turn multiple statements into a single compound statement

The `break` statement may or may not appear after each statement

`switch`, `case`, `break`, and `default` are reserved words

Consider the following statements, where grade is a variable of type char:

```
switch (grade)
{
case 'A':
    cout << "The grade is 4.0.";
    break;
case 'B':
    cout << "The grade is 3.0.";
    break;
case 'C':
    cout << "The grade is 2.0.";
    break;
case 'D':
    cout << "The grade is 1.0.";
    break;
case 'F':
    cout << "The grade is 0.0.";
    break;
default:
    cout << "The grade is invalid.";
}
```

In this example, the expression in the switch statement is a variable identifier. The variable grade is of type char, which is an integral type. The possible values of grade are 'A', 'B', 'C', 'D', and 'F'. Each case label specifies a different action to take, depending on the value of grade. If the value of grade is 'A', the output is:

```
The grade is 4.0.
```

```
#include <iostream>
using namespace std;
int main(){
int firstnum,secondnum,option;
cin >> option>>firstnum>>secondnum;
switch(option){
case 1:
cout<<firstnum+secondnum<<endl;
break;
case 2:
cout<<firstnum-secondnum<<endl;
break;
case 3:
cout<<firstnum*secondnum<<endl;
break;
case 4:
cout<<firstnum/secondnum<<endl;
break;
default:
cout<<"option is incorrect";
break;  }
return 0;  }
```