Programming Language

Fundamentals *(Day 1 Part 1)*

By: Bhaskar Ghosh
bjghosh@gmail.com

Updated: 9 August 2024

C++ Programming Language

Fundamentals (Day 1 Part 1)

- Origin of C++
- Installing C++ and C++ Editor/IDE
- Best Practices

- Comments
- C++ Tokens
- Keywords
- Identifiers
- Constants
- Datatypes
- Type Modifiers
- Type Conversion
- Operators
- Basic Input/Output

C++ Programming Language

Origin

- C++ is a general-purpose, high-level programming language developed in 1979 by Bjarne Stroustrup at AT & T Bell Laboratories.
- C++ was created as superset of the C programming language.
- Initially referred to as "new C".
- Then it was renamed to "C with Classes".
- Finally named “C++” by Rick Mascitti.

Language Name	Developed By	Year of Origin
Algol	International Group	1960
Basic Combined Programming Language	Martin Richards	1967
B	Ken Thompson	1970
C	Dennis Ritchie	1972
K&R C	Brian Kernighan & Dennis Ritchie	1978
C++	Bjarne Stroustrup	1980

C++ Programming Language

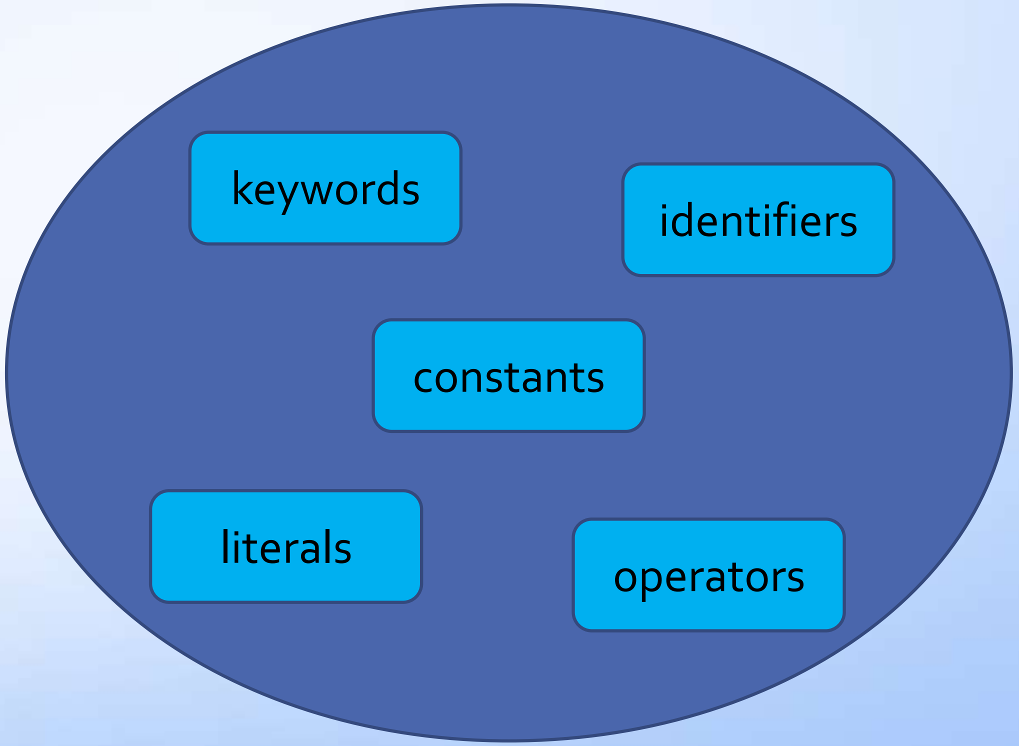
Installing GNU C++ Compiler and VSCode editor

- MingW and VSCode
  - https://code.visualstudio.com/docs/languages/cpp
  - https://code.visualstudio.com/docs/cpp/config-mingw#_prerequisites
  - https://github.com/niXman/mingw-builds-binaries/releases
  - https://stackoverflow.com/questions/75271199/changing-c-compiler-version-on-vs-code
- Winlibs and Code::Blocks
  - https://winlibs.com/#usage-codeblocks
  - https://www.codeblocks.org/downloads/
  - https://winlibs.com/

C++ Programming Language

Tokens

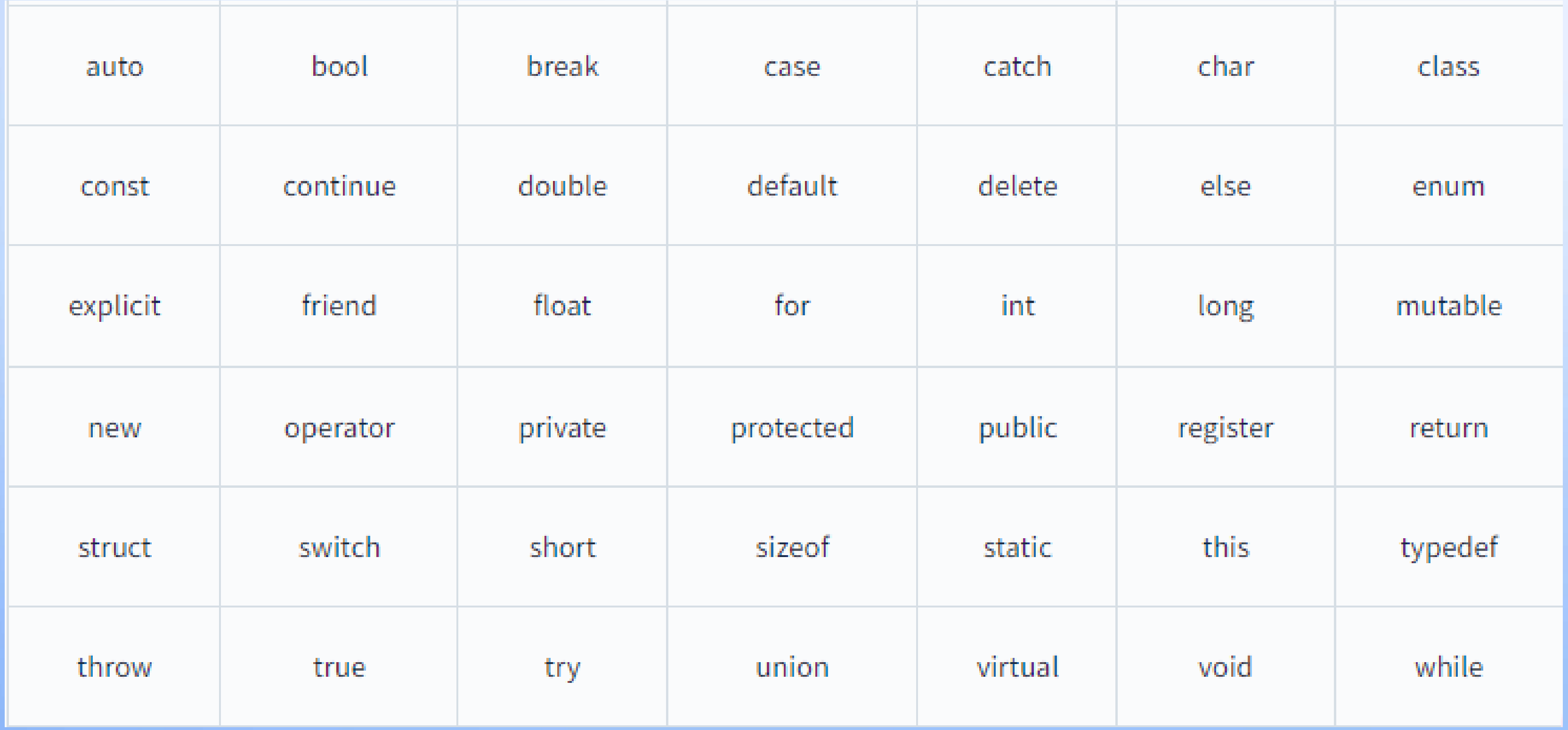
- Like living cells in the human body are the smallest possible units of life, tokens in C++ are the smallest building blocks of a C++ program.



C++ Programming Language

Keywords

- Keywords are predefined words that have special meanings to the compiler.
- There are a total of 95 Keywords in C++. All are lowercase (case-sensitive).



auto	bool	break	case	catch	char	class
const	continue	double	default	delete	else	enum
explicit	friend	float	for	int	long	mutable
new	operator	private	protected	public	register	return
struct	switch	short	sizeof	static	this	typedef
throw	true	try	union	virtual	void	while

C++ Programming Language

Identifiers

- Identifiers refer to the unique names of variables, functions, arrays, classes, etc. created by the programmer.
- Rules for naming identifiers
  - Identifiers can be composed of letters, digits, and the underscore character.
  - It has no limit on name length.
  - It must begin with either a letter or an underscore, i.e. it can not start with a digit or any special character.
  - It is case-sensitive.
  - We cannot use keywords as identifiers.

C++ Programming Language

Best Practices

- Naming Conventions
  - Class name should be a noun, and begin with uppercase letter.
  - Variable name should begin with lower case letter.
  - Method name should be a verb, and begin with a lower case letter.
  - Upper case letters should be used as word separators, and lower case for the rest of the word.
  - Better not to use underscores in names of identifiers, except for use in naming constants.
  - Constants should be named all in upper case letters with underscores as word separators.
- Comments
  - Used for code explanation, documentation, debugging aid, maintainability, code review and collaboration, to-do and future work.

  -
    ```
    /* multi line
       comment (C style)
    */
    ```

C++ Programming Language

Variables

- A variable is a container (storage area) to hold data.
- To indicate the storage area, each variable should be given a unique name (identifier).
- In C++, all the variables must be declared before use.
- Declaring/Initialising a variable:
  - *type variable_name;*
  - *type variable_name = value;*
- Example:

```
int age = 14;
```

C++ Programming Language

Constants

- In C++, we can create variables whose value cannot be changed.
- For that, we use the `const` keyword.
  - *const type CONSTANT_NAME = value;*
- Like in C language, a constant can also be created using the `#define` preprocessor directive.
- Example:

```
const float PI = 3.14;
PI = 6.28; //error: PI is a constant, hence can't change its value
```

C++ Programming Language

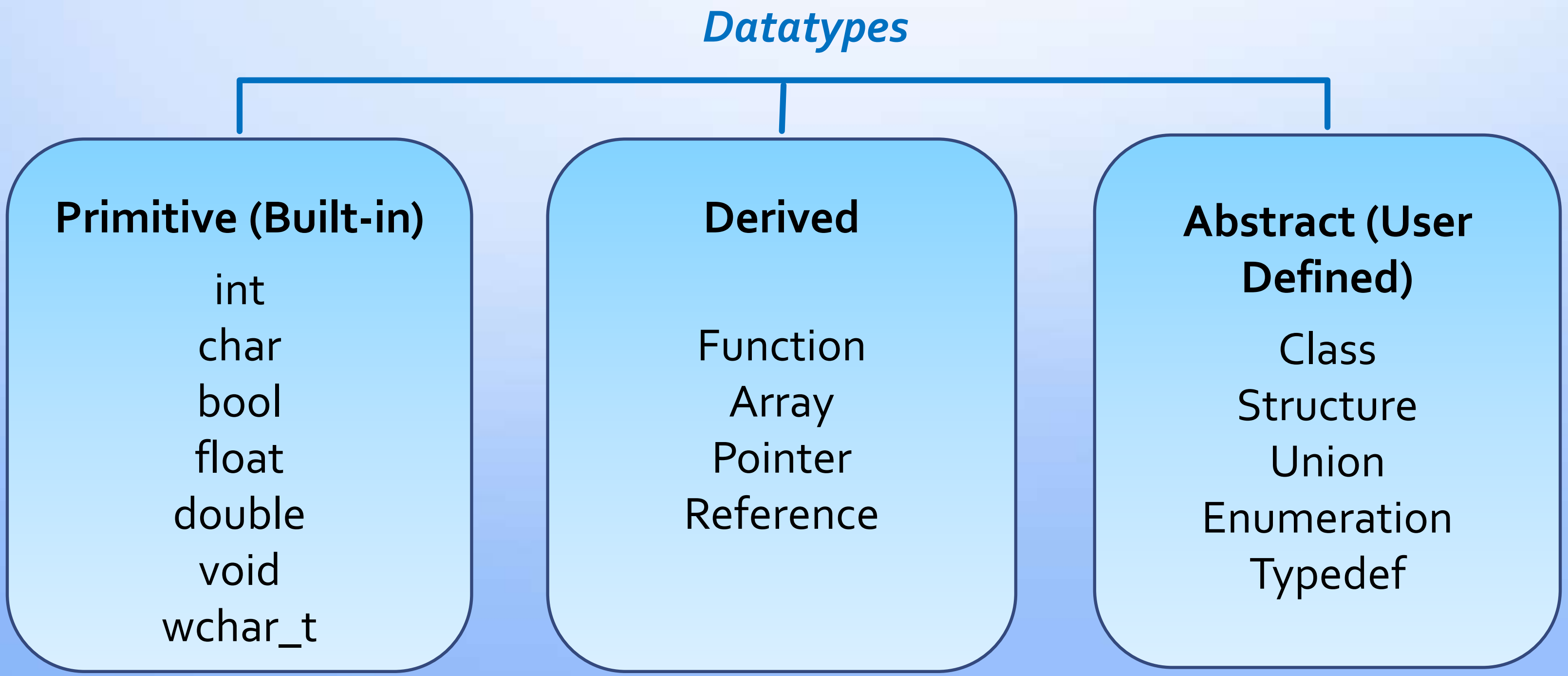
Literals

- Literals are data used for representing fixed values. They can be used directly in the code.
- Types of literals
  - Integers (11, 021, 0x7f, etc.)
  - Floating points (2.0, -4.666, 0.22E-5, etc.)
  - Characters ('a', 'K', '1', '[', etc.)
  - Strings ("wow", "", " ", "a", etc.)

C++ Programming Language

Datatypes

- In C++, data types are declarations for variables.
- They determine the type and size of data associated with variables.
  - example: `int height = 165;`



Datatypes

Primitive (Built-in)

int
char
bool
float
double
void
wchar_t

Derived

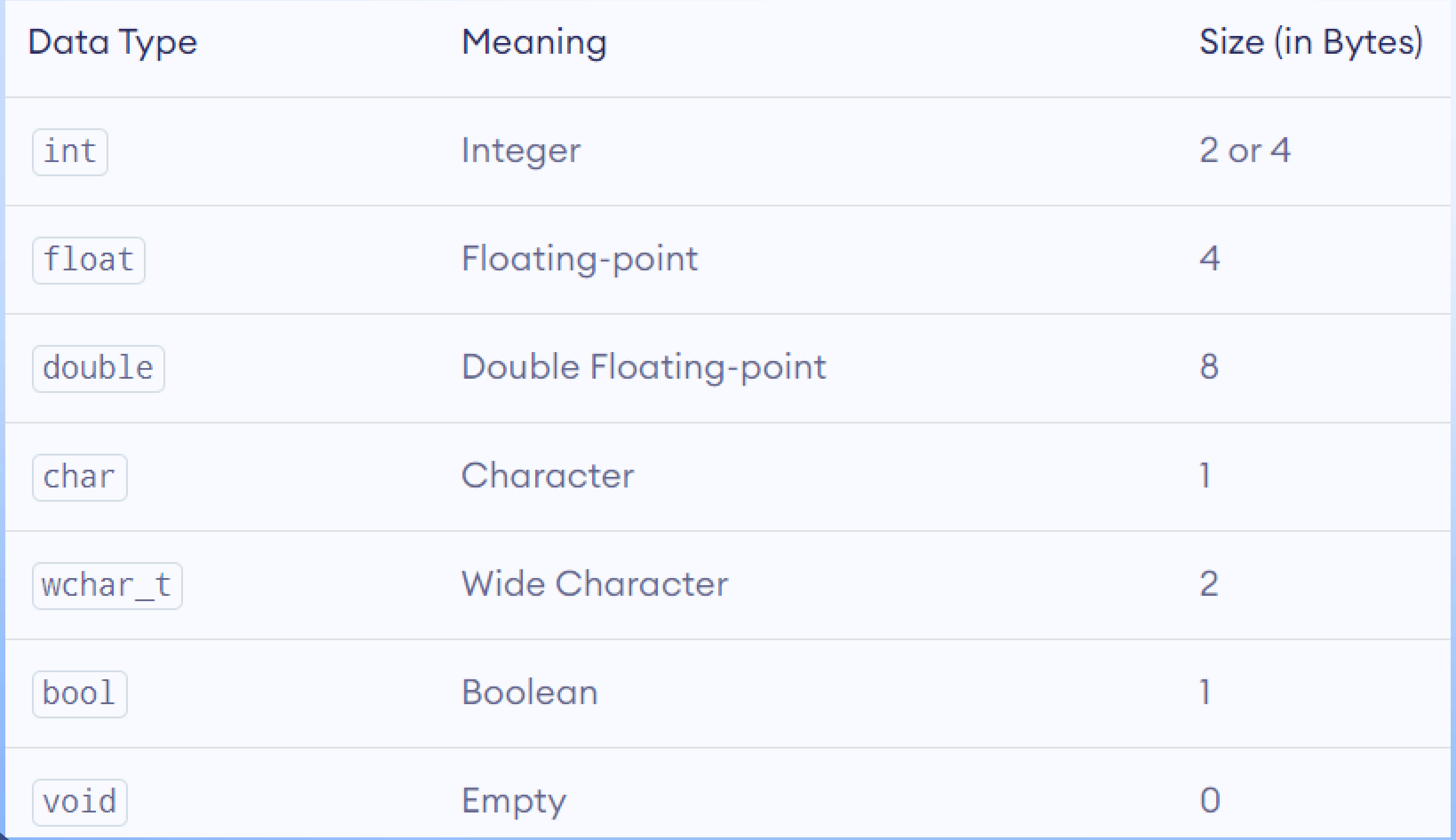
Function
Array
Pointer
Reference

Abstract (User Defined)

Class
Structure
Union
Enumeration
Typedef

C++ Programming Language

Datatypes



Data Type	Meaning	Size (in Bytes)
`int`	Integer	2 or 4
`float`	Floating-point	4
`double`	Double Floating-point	8
`char`	Character	1
`wchar_t`	Wide Character	2
`bool`	Boolean	1
`void`	Empty	0

C++ Programming Language

Type Modifiers

- We can further modify some of the fundamental data types by using type modifiers.
- There are 4 type modifiers in C++.
  - signed
  - unsigned
  - short
  - long

C++ Programming Language

Type Modifiers

DataType	Size(in Bytes)	Range
int or signed int	**4 Bytes**	**-2,147,483,648 to 2,147,483,647**
unsigned int	4 Bytes	0 to 4,294,967,295
short int	**2 Bytes**	**-32,768 to 32,767**
long int	4 Bytes	-2,147,483,648 to 2,147,483,647
unsigned short int	**2 Bytes**	**0 to 65,535**
unsigned long int	8 Bytes	0 to 4,294,967,295
long long int	**8 Bytes**	**-(2^63) to (2^63)-1**
unsigned long long int	8 Bytes	0 to 18,446,744,073,709,551,615
signed char	**1 Bytes**	**-128 to 127**
unsigned char	1 Bytes	0 to 255
wchar_t	**2 or 4 Bytes**	**1 wide character**
float	4 Bytes	
double	**8 Bytes**	
long double	12 Bytes	

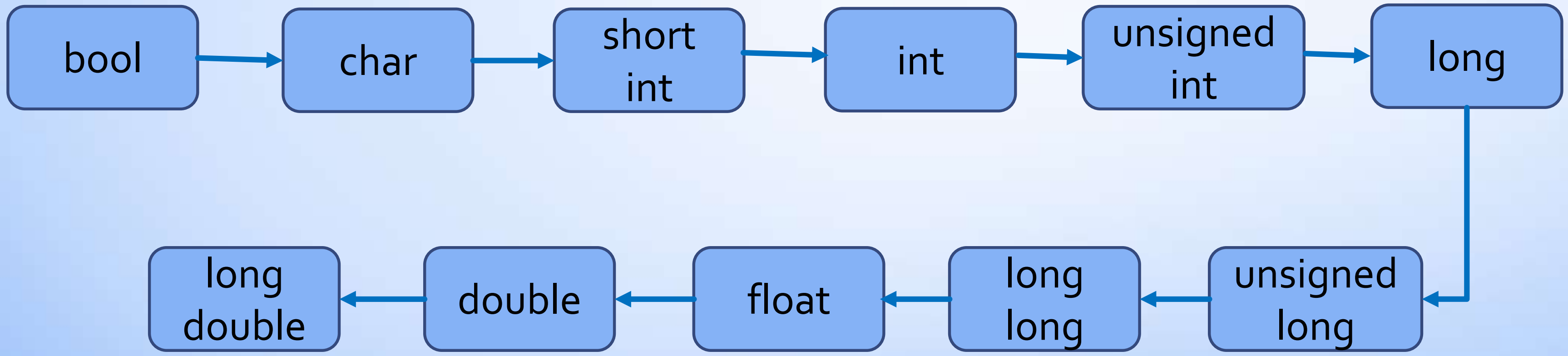
C++ Programming Language

Type Conversion

- The conversion of a variable from one data type to another.
- Most commonly used to perform mathematical and logical operations on two variables with different data types.
- ***Implicit type conversion***
  - Done automatically by the compiler.
  - It does not require any effort from the programmer.
  - All the data types of the variables are upgraded to the data type of the variable with largest data type (type promotion).
- ***Explicit type conversion***
  - Also called type casting and it is user-defined.
  - We can typecast (change) the data type of a variable to another type if we do not want to follow the implicit type conversion rules.

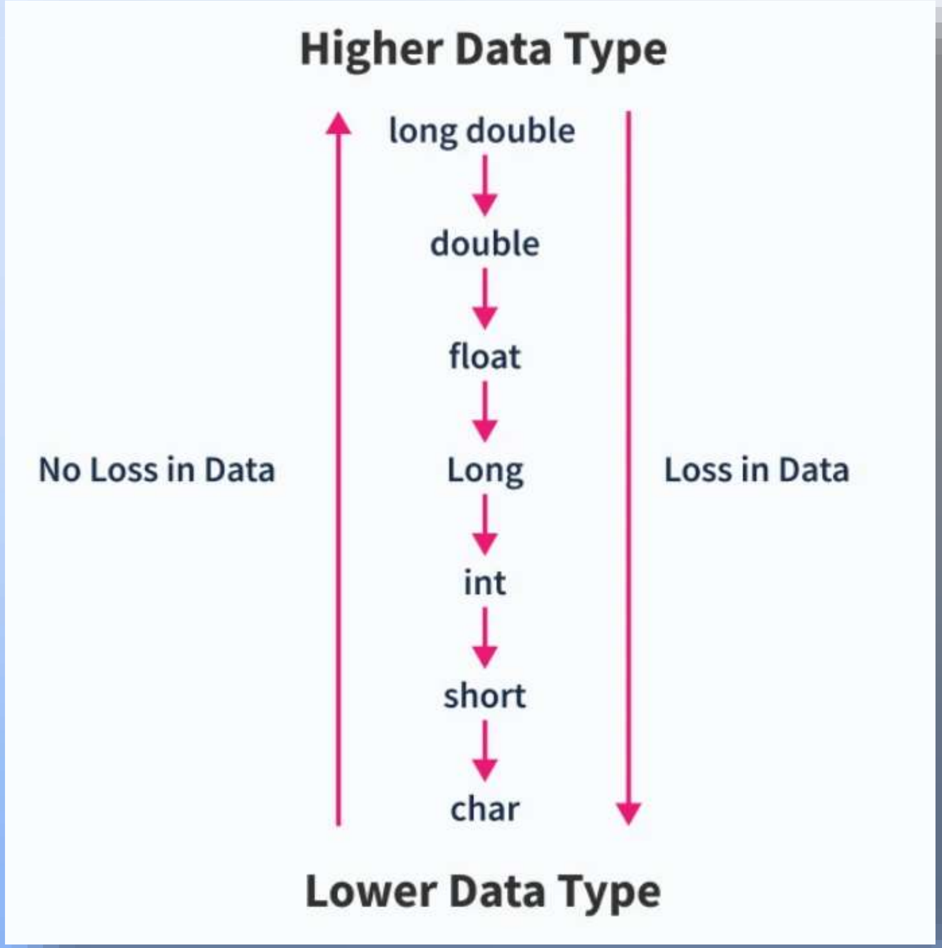
C++ Programming Language

Implicit Type Conversion



C++ Programming Language

Implicit Type Conversion



Higher Data Type

long double

double

float

Long

int

short

char

No Loss in Data

Loss in Data

Lower Data Type

C++ Programming Language

Explicit Type Conversion – Conversion Using the Assignment Operator

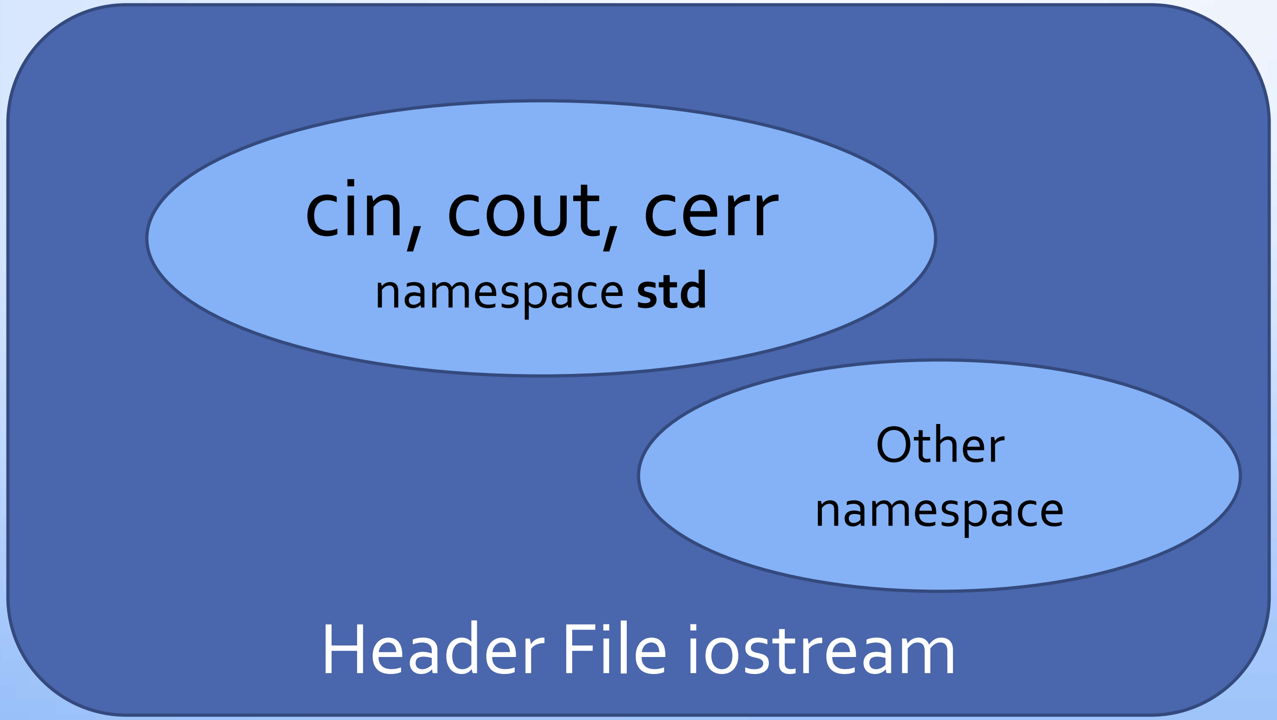
- Also referred to as a forced casting.
- Done by explicitly declaring the required data type in front of the expression.
- Can be done in two ways:
  - ***C-style type casting***
    - Usually used in the C programming language.
    - Also known as cast notation.
    - Syntax: (datatype)expression;
  - ***Function-style casting***
    - Also known as old C++-style type casting.
    - Syntax: datatype(expression);

```
char char_var = 'a';
int int_var;
int_var = (int) char_var;
```

```
int_var = int (char_var);
```

C++ Programming Language

Header Files and Namespaces: Concept



cin, cout, cerr

namespace **std**

Other namespace

Header File iostream

C++ Programming Language

Explicit Type Conversion – Conversion Using the Cast Operator

- 4 types of Casting in C++ programming language:
  - ***Static Cast***
    - Can perform all the conversions that are done implicitly.
    - Done at compile time.
  - ***Dynamic Cast***
    - Done at run-time.
    - Used to check the validity of a cast.
  - Const Cast
  - Reinterpret Cast

C++ Programming Language

Explicit Type Conversion – Conversion Using the Cast Operator

- ***Static Cast***
  - Can perform
    - conversions between the pointers of classes related to each other.
    - upcast (conversion from a derived class to a base class) operations
    - downcast (conversion from a base class to a derived class) operations.
  - Syntax: `static_cast <datatype> (expression)`

```
double num = 3.7 * 5.5;
int cast_var;
cast_var = static_cast <int> (num);
```

C++ Programming Language

Explicit Type Conversion – Conversion Using the Cast Operator

- ***Dynamic Cast***
  - Can only be used with pointers and references to classes (or void*).
  - Can only be used when we typecast from a parent class to a derived class.
  - If the conversion is not possible,
    - it returns a null pointer (for pointer conversions) or
    - throws a bad_cast exception (for reference conversions).
  - Uses the Run-Time Type Identification (RTTI) mechanism
    - to make all information about the data type of an object available at the run-time.
  - Syntax: `dynamic_cast <datatype> (expression)`
  - More about this after the section on “Classes and Objects”.

C++ Programming Language

Explicit Type Conversion – Conversion Using the Cast Operator

- ***Const Cast***
  - Used to change an object's constant value or to remove the constant nature of any object.
  - Used to modify the const or volatile qualifier of a variable.
  - Generally used in programs with one or more objects with some constant value(s) that need to be changed at some point in the program.
  - For a const cast operation to be successful, the pointer and the source being cast should be of the same data.
    - to make all information about the data type of an object available at the run-time.
  - Syntax: `const_cast <datatype> (expression)`
  - More about this after the section on "Pointers".

C++ Programming Language

Explicit Type Conversion – Conversion Using the Cast Operator

- ***Reinterpret Cast***
  - Used to convert one pointer type to another, regardless of whether the classes are related.
  - Does not check whether the pointer type and data pointed out by the pointer are the same.
  - Syntax: `reinterpret_cast <datatype> (expression)`
  - More about this after the section on “Pointers”.
- Note:
  - `const_cast` and `reinterpret_cast` are generally not reccommended as they vulnerable to different kinds of errors.

C++ Programming Language

Operators

- Used to perform specific mathematical or logical computation.
- An operator operates the operands.
  - e.g.
    int sum = a + b;
  - **a++, --a**

	Operator	Type of Operator
Unary	++, --	Increment, Decrement
Binary	+, -, *, /, %	Arithmetic
Binary	<, >, <=, >=, ==, !=	Relational
Binary	&&, ||, !	Logical
Binary	&, |, <<, >>, ~, ^	Bitwise
Binary	=, +=, -=, *=, /=, %=	Assignment
Ternary	?:	Conditional

C++ Programming Language

Operator Precedence (1)

Precedence	Operator	Description	Associativity
1	::	Scope Resolution	Left to Right
2	a++ a-- type( ) type{ } a( ) a[ ] . ->	Suffix/postfix increment Suffix/postfix decrement Function cast Function cast Function call Subscript Member access from an object Member access from object ptr	Left to Right

C++ Programming Language

Operator Precedence (2)

Precedence	Operator	Description	Associativity
3	++a --a +a -a ! ~ (type) *a &a sizeof co_await new new[ ] delete delete[]	Prefix increment Prefix decrement Unary plus Unary minus Logical NOT Bitwise NOT C style cast Indirection (dereference) Address-of Size-of await-expression Dynamic memory allocation Dynamic memory deallocation	Right to Left

C++ Programming Language

Operator Precedence (3)

Precedence	Operator	Description	Associativity
4	.* ->*	Member object selector Member pointer selector	Left to Right
5	a * b a / b a % b	Multiplication Division Modulus	Left to Right
6	a + b a - b	Addition Subtraction	Left to Right
7	<< >>	Bitwise left shift Bitwise right shift	Left to Right
8	<=<	Three-way comparison operator	Left to Right

C++ Programming Language

Operator Precedence (4)

Precedence	Operator	Description	Associativity
9	< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left to Right
10	== !=	Equal to Not equal to	Left to Right
11	&	Bitwise AND	Left to Right
12	^	Bitwise XOR	Left to Right
13	|	Bitwise OR	Left to Right
14	&&	Logical AND	Left to Right
15	||	Logical OR	Left to Right

C++ Programming Language

Operator Precedence (5)

Precedence	Operator	Description	Associativity
16	a ? b : c throw co_yield = += -= *= /= %= <<= >>= &= ^= |=	Ternary Conditional throw operator yield expression (C++ 20) Assignment Addition Assignment Subtraction Assignment Multiplication Assignment Division Assignment Modulus Assignment Bitwise Shift Left Assignment Bitwise Shift Right Assignment Bitwise AND Assignment Bitwise XOR Assignment Bitwise OR Assignment	Right to Left
17	,	Comma operator	Left to Right

C++ Programming Language

Basic Input / Output

- In C++ input and output are performed in the form of a sequence of bytes or more commonly known as streams.
- *Input Stream*
  - If the direction of flow of bytes is from the device (for example, Keyboard) to the main memory then this process is called input.
- *Output Stream*
  - If the direction of flow of bytes is opposite, i.e. from main memory to device (display screen) then this process is called output.

C++ Programming Language

Basic Input / Output

- Standard Input (cin)
  - `cin >> a;`
- Standard Output (cout)
  - `cout << "a is " << a;`
- Un-buffered Standard Error (cerr)
  - `cerr << "Error Occurred";`
  - Used when one needs to display the error message immediately.
  - We can not redirect **cerr** outputs to a file, since it is un-buffered stream.
- Buffered Standard Error (clog)
  - Error is first inserted into a buffer and remains in the buffer until it is not fully filled, or the buffer is not explicitly flushed (using flush()).
- Header Files required
  - **iostream:** Contains definitions of objects like cin, cout, cerr, clog, etc.

    ```
    #include <iostream>
    using namespace std;
    ```
  - **iomanip:** for manipulating streams (setw, setprecision, etc.)
  - **fstream:** file stream handling.

Programming Language

Flow Control *(Day 1 Part 2)*

By: Bhaskar Ghosh

bjghosh@gmail.com

Updated: 9 August 2024

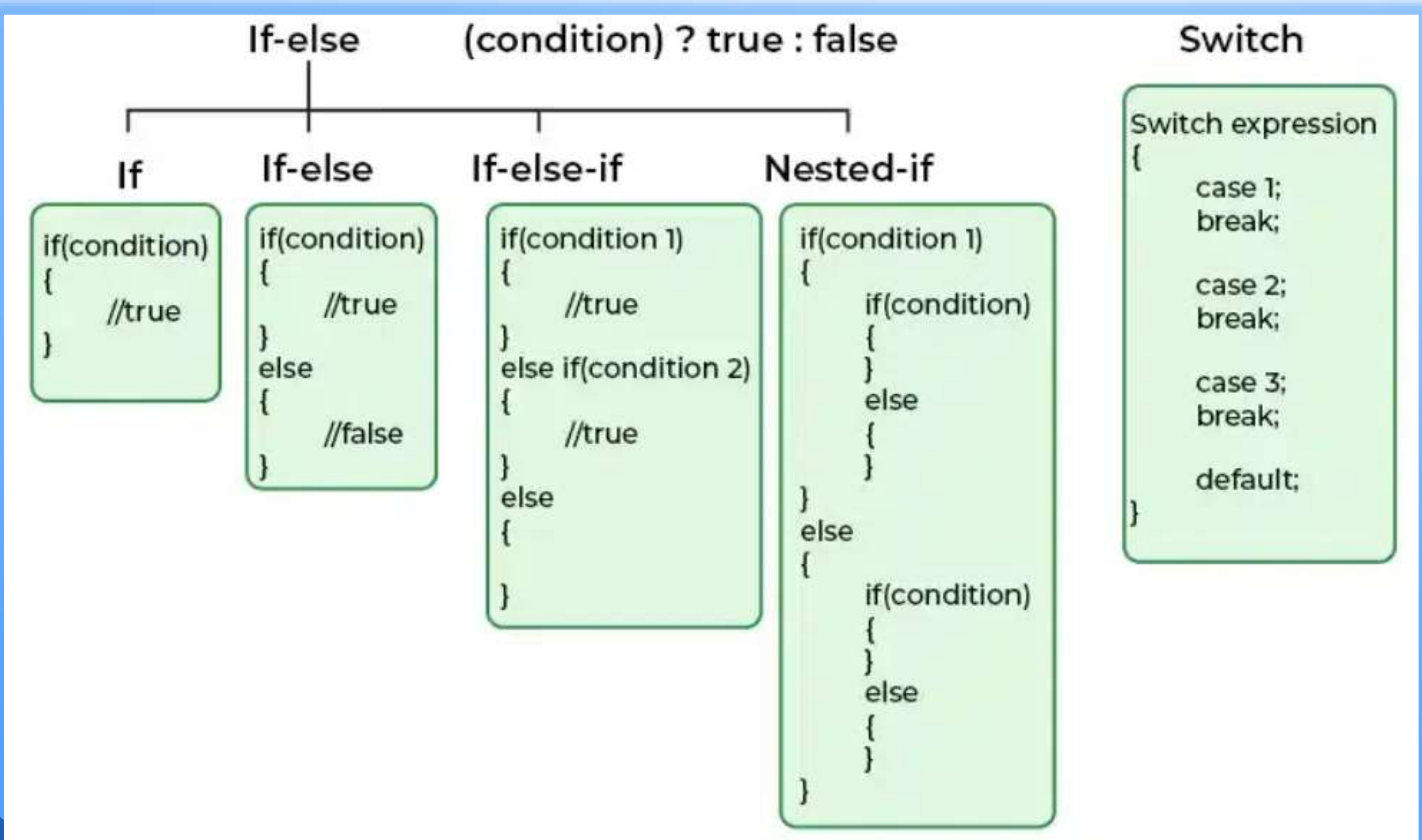
C++ Programming Language

Flow Control (Day 1 Part 2)

- Conditional Statements
  - if…else and nested if…else
  - ternary operator
  - switch…case
- Iteration (C++ loops – for, ranged-for, while, do while)
- Jump Statements (break, continue, return)
- Problem Solving using C++ [Level 1]
  - Logic Building and Debugging

C++ Programming Language

Conditional Statements



If-else

(condition) ? true : false

Switch

If

```
if(condition)
{
        //true
}
```

If-else

```
if(condition)
{
        //true
}
else
{
        //false
}
```

If-else-if

```
if(condition 1)
{
        //true
}
else if(condition 2)
{
        //true
}
else
{

}
```

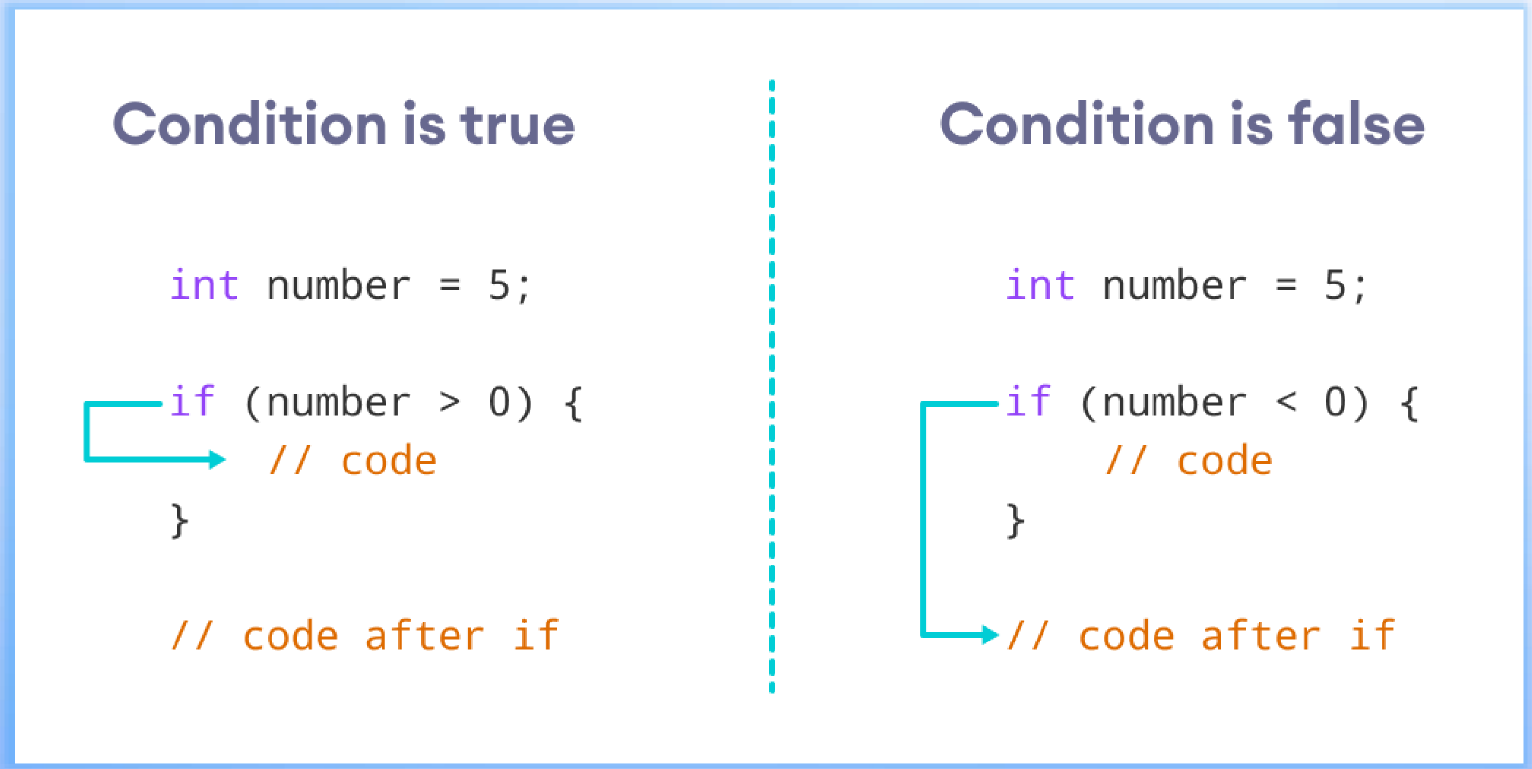
Nested-if

```
if(condition 1)
{
        if(condition)
        {
        }
        else
        {
        }
}
else
{
        if(condition)
        {
        }
        else
        {
        }
}
```

```
Switch expression
{
        case 1;
        break;

        case 2;
        break;

        case 3;
        break;

        default;
}
```

C++ Programming Language

Conditional Statements – if



Condition is true

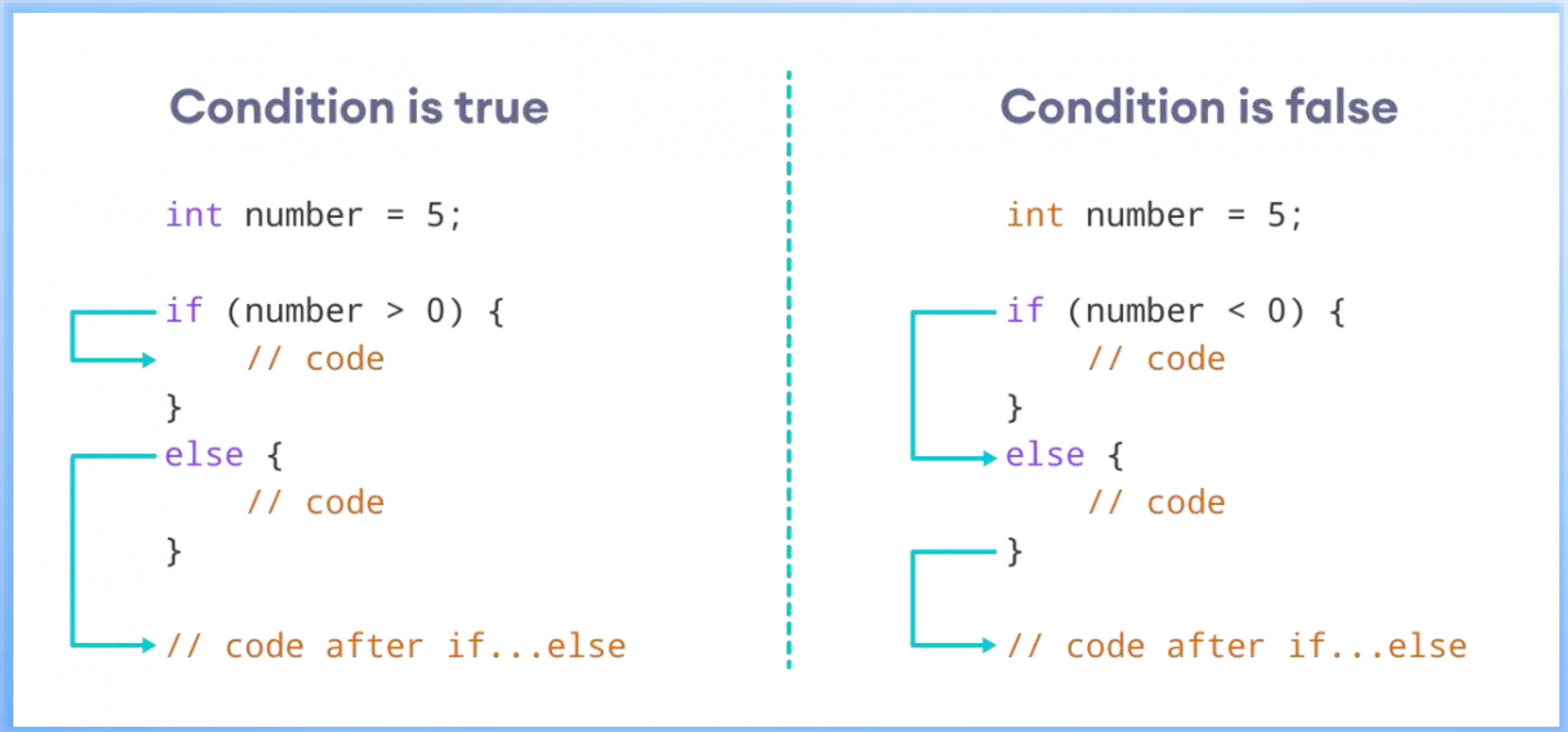
```
int number = 5;

if (number > 0) {
    // code
}

// code after if
```

Condition is false

```
int number = 5;

if (number < 0) {
    // code
}

// code after if
```

C++ Programming Language

Conditional Statements – if…else



Condition is true

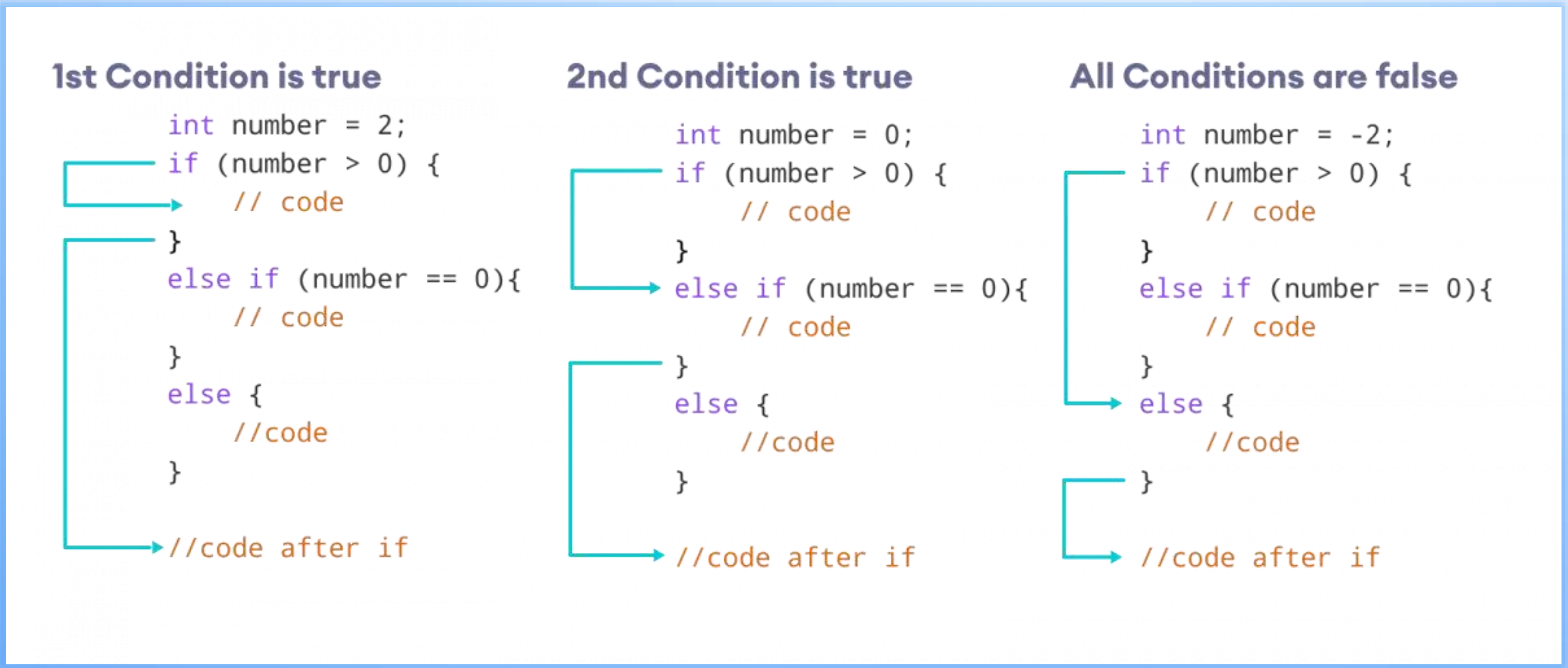
```
int number = 5;

if (number > 0) {
    // code
}
else {
    // code
}

// code after if...else
```

Condition is false

```
int number = 5;

if (number < 0) {
    // code
}
else {
    // code
}

// code after if...else
```

C++ Programming Language

Conditional Statements – if…else…else…if



1st Condition is true

```
int number = 2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

2nd Condition is true

```
int number = 0;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

All Conditions are false

```
int number = -2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

C++ Programming Language

Conditional Statements – Ternary Operator ?:

- A concise, inline method that evaluates the test condition and executes an expression out of two based on the result of the condition..
- Also called the conditional operator.
- Syntax
  - `condition ? expression1 : expression2;`
- Example
  - `string result = (marks >= 40) ? "passed" : "failed";`

C++ Programming Language

Conditional Statements – switch…case

- Allows us to execute a block of code among many alternatives.
- Syntax

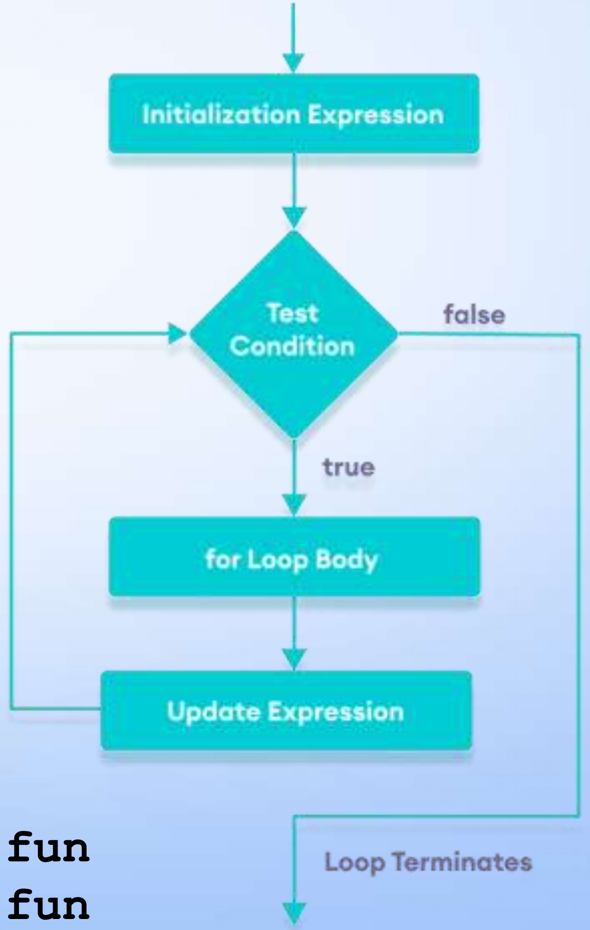
  -
    ```
    switch (expression)   {
     case constant1:// code to be executed if expression is equal to constant1
          break;

     case constant2:  // code to be executed if expression is equal to constant2;

              break;

     default:  // code to be executed if expression doesn't match any constant
    }
    ```
- expression is evaluated once and compared with the values of each case label.
- If the expression is equal to a case constant, the code after that case constant is executed until a break is encountered.

If there is no match, the code after default: is executed.

C++ Programming Language

Iteration – for loop

- Loops are used to repeat a block of code
- In C++, there are three types of loops
  - for loop
    - Enhanced for-each loop
  - while loop
  - do...while loop



```
for (initialExpr; testExpr; updateExpr) {
    // body of the loop
}
```

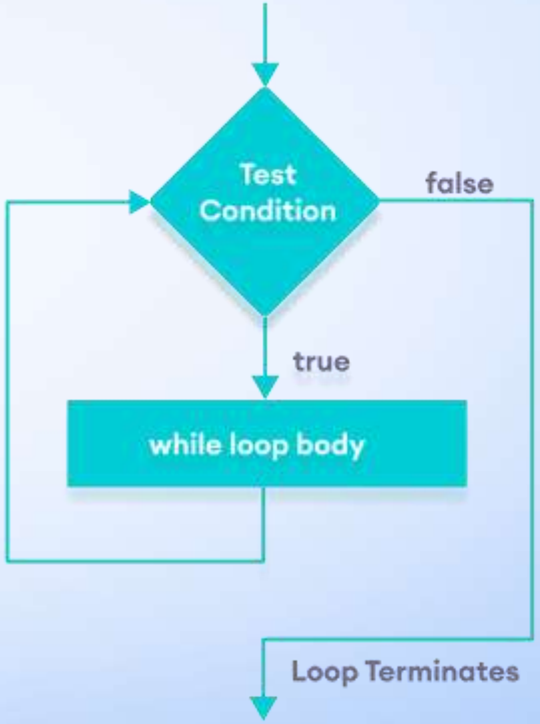
```
int n = 5;
for (int i = 1; i <= n; ++i) {
    cout << "C++ is fun" << endl;
}
```

```
C++ is fun
C++ is fun
C++ is fun
C++ is fun
C++ is fun
```

C++ Programming Language

Iteration – while loop

```
while (testExpression) {
    // body of loop
}
```



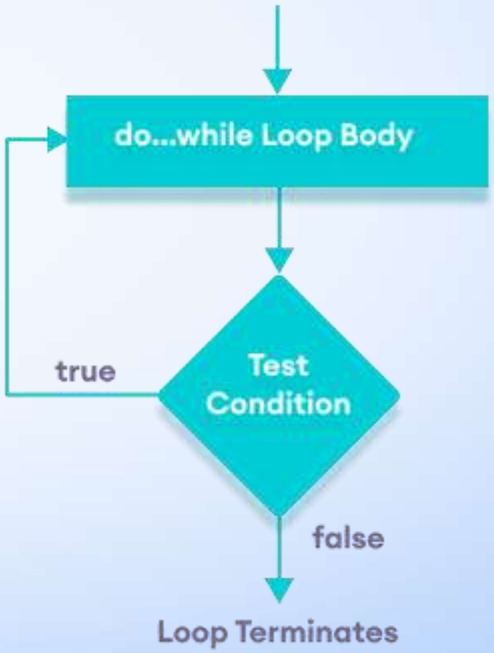
```
int i = 1, n = 5;
while (i <= n) {
  cout << i << endl;
}
```

```
1
2
3
4
5
```

C++ Programming Language

Iteration – do…while loop

```
do {
    // body of loop
} while (testExpression);
```



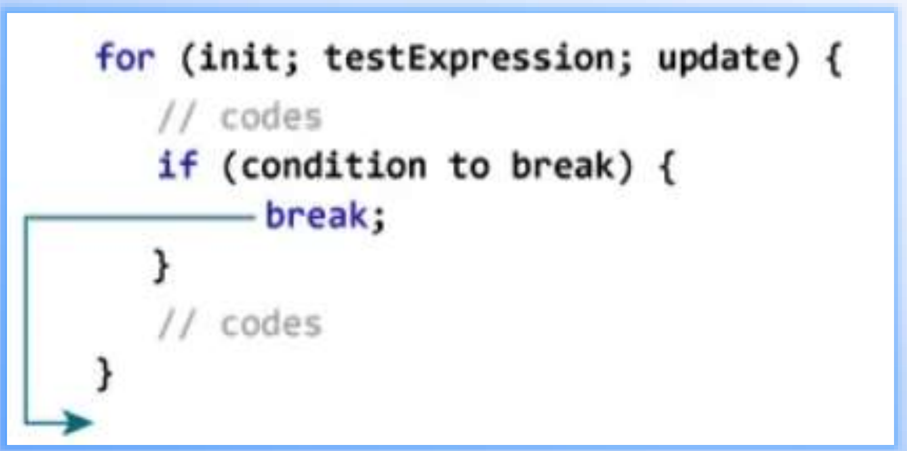
```
int i = 1, n = 5;
do {
      cout << i << endl;
} while (i <= n);
```

```
1
2
3
4
5
```

C++ Programming Language

Jump statements – break

- With the break statement we can stop the loop even if the condition (test expression) is true.



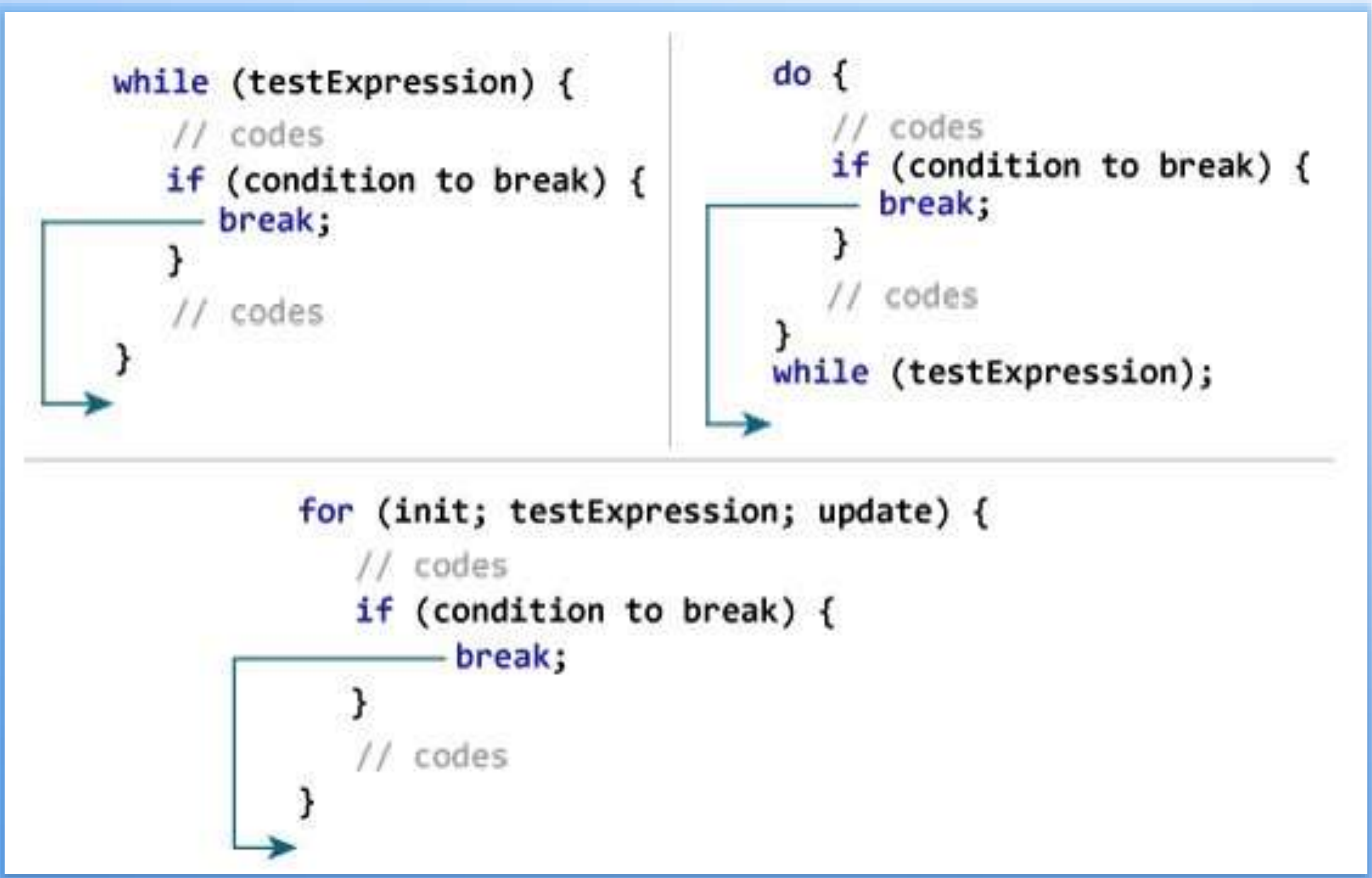
```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
int n = 5;
for (int i = 1; i <= n; ++i) {
        if (i == 3)
                break;
    cout << "C++ is fun:" << i << endl;
}
```

```
C++ is fun:1
C++ is fun:2
```

C++ Programming Language

Jump statements – break



```
while (testExpression) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

```
do {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
   // codes
   if (condition to break) {
      break;
   }
   // codes
}
```

C++ Programming Language

Jump statements – continue

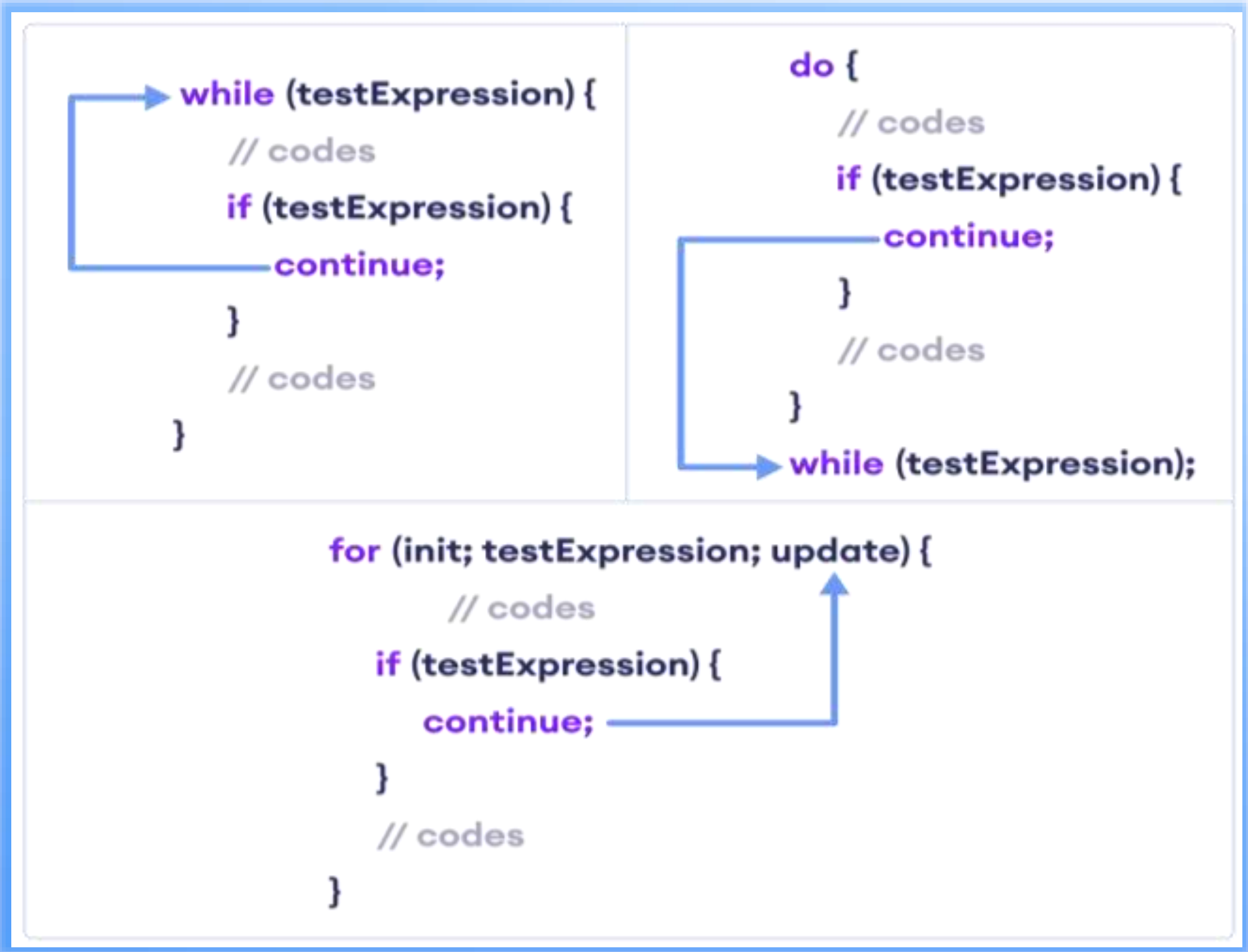
- The continue statement skips the current iteration of a loop (for, while, do…while, etc).
- After the continue statement, the program moves to the end of the loop.

```
int n = 5;
for (int i = 1; i <= n; ++i) {
        if (i == 3)
                continue;
    cout << "C++ is fun:" << i << endl;
}
```

```
C++ is fun:1
C++ is fun:2
C++ is fun:3
C++ is fun:4
```

C++ Programming Language

Jump statements – continue



```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
        // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

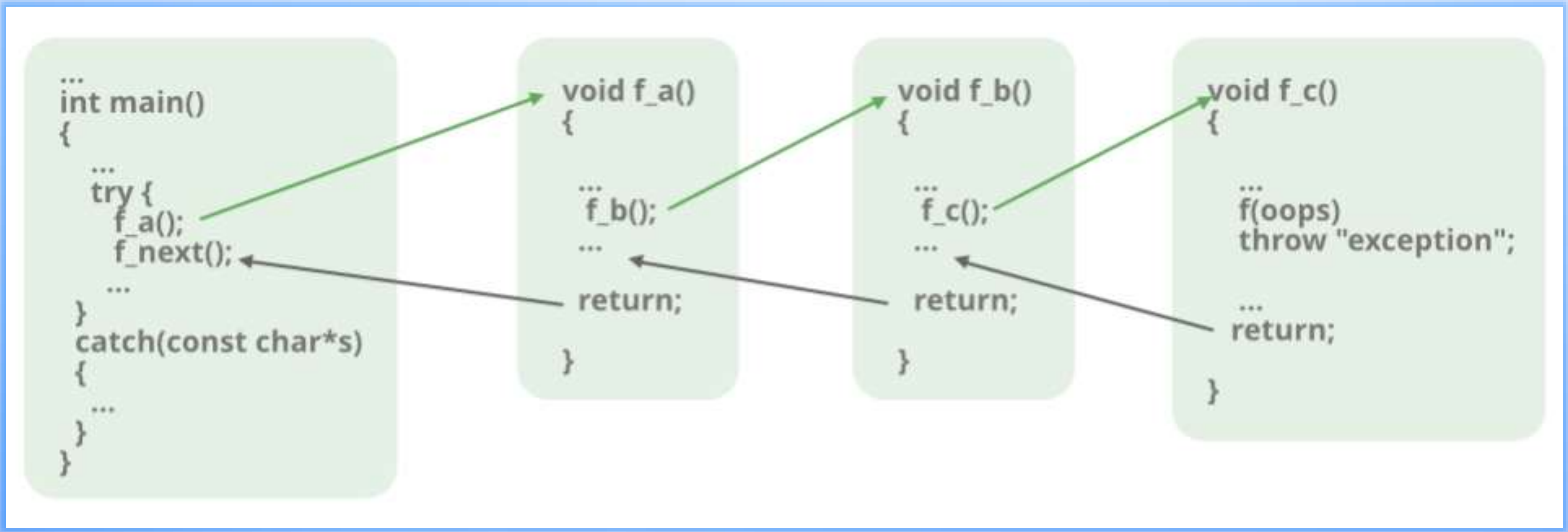
C++ Programming Language

Jump statements – return

- The return statement can be used in any part of a function.
- Used to stop the execution of the current function and transfer the program control to the point from where it has been called.
- If a return statement is used in the main function, it will stop the compilation process of the program.
- It is also used to transfer a computed value of the function to the variable used to call the function.
- Every function has a return statement with some returning value except the void() function.
- But, void() function can also have the return statement to end the execution of the function.

C++ Programming Language

Jump statements – return



```
...
int main()
{
   ...
   try {
     f_a();
     f_next();
    ...
  }
  catch(const char*s)
  {
   ...
  }
}
```

```
void f_a()
{

   ...
   f_b();
   ...

   return;

}
```

```
void f_b()
{

   ...
   f_c();
   ...

   return;

}
```

```
void f_c()
{

   ...
   f(oops)
   throw "exception";

   ...
   return;

}
```

C++ Programming Language

Problem Solving using C++ [Day 1]

logic Building and Debugging

C++ Programming Language

Problem Solving using C++ (1.1)

1. WAP to calculate sum, difference, multiplication, division of 2 numbers, based on user choice.
2. WAP to print DISTINCTION if marks >= 75, PASS if marks >= 60, else FAIL, using (a) if..else..if and (b) ?: operator.
3. WAP to ask name of the user, and greet the user using the name, until the user types BYE.
4. WAP to swap 2 numbers.
5. WAP to find out if a given year is a leap year or not.
6. WAP to input a number from the user and print its reverse value as a number, without using arrays.

C++ Programming Language

Problem Solving using C++ (1.2)

8. Write an entire C++ program that reads a positive integer entered by an interactive user and then prints out all the positive divisors of that integer in a column and in decreasing order. The program should allow the user to repeat this process as many times as the user likes.

Also provide a way for the user to end the program.

9. WAP to print the calendar of the year and the month inputted by the user.