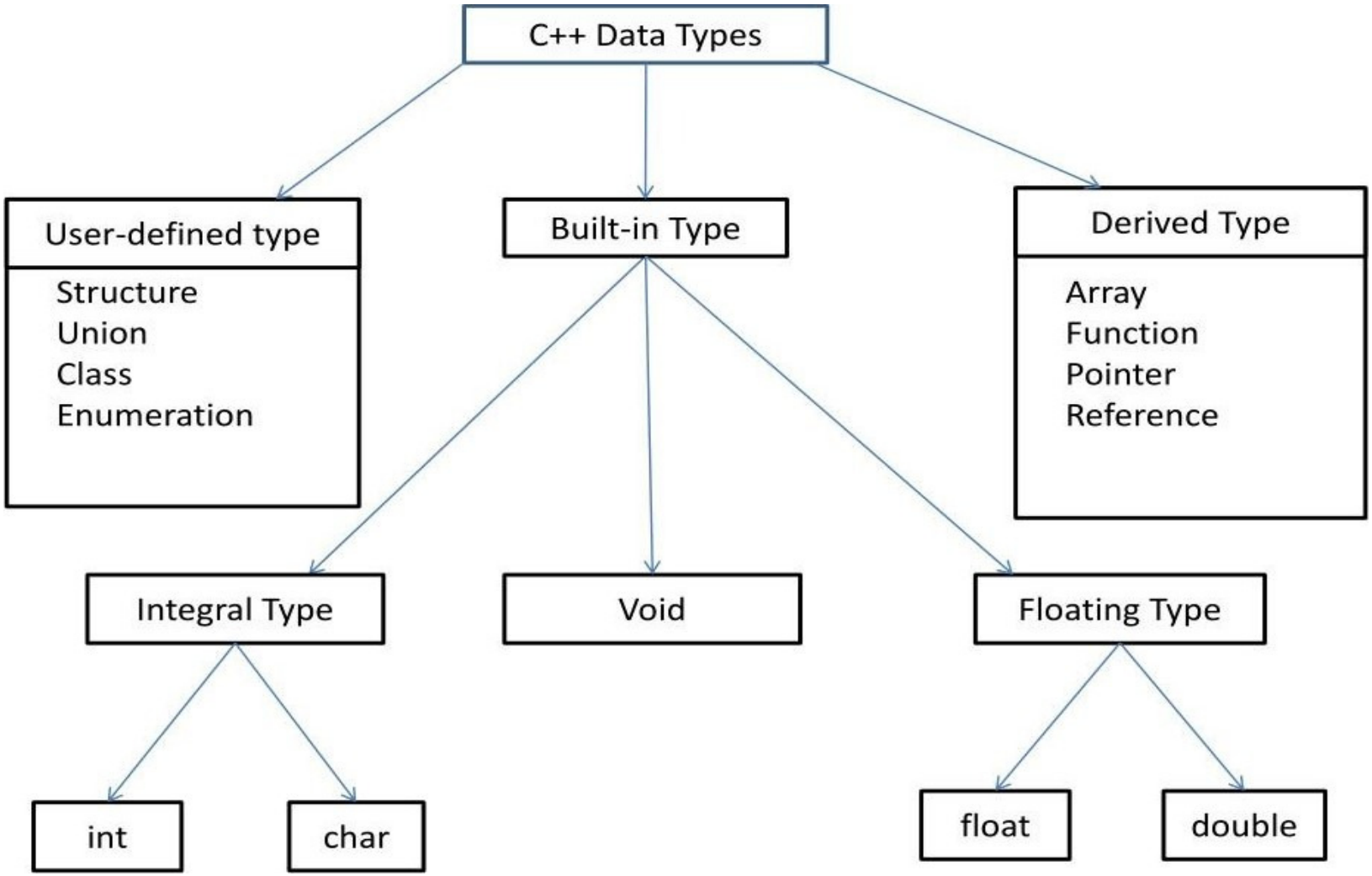
Object Oriented Programming using C++

Data Types

- Data is the smallest unit of information that a computer processes.
- The data type defines the kind of data being manipulated.
- Basic data types include:
    - Integer: Represents whole numbers.
    - Floating-point: Represents numbers with decimal points.
    - Character: Represents individual characters.
- C++ supports both basic and derived data types.
- ANSI C++ introduced two additional data types:
    - `bool`: Represents boolean values (`true` or `false`).
    - `wchar_t`: Represents wide characters, accommodating a larger range of characters beyond standard ASCII.


C++ Data Types
User-defined type
Structure
Union
Class
Enumeration
Built-in Type
Derived Type
Array
Function
Pointer
Reference
Integral Type
Void
Floating Type
int
char
float
double

Primary/ Fundamental Data type

—There are five basic data types:

char

int

float

double

void

—ANSI C++ adds two more:

bool Boolean value

wchar t wide character

wchar_t Type Specifier

- The `wchar_t` type specifier is an integral type designed to store wide character literals.

- A wide character literal is a character literal prefixed with the letter `L`, such as `L'x'`.

Integer Data Type

- Integer data type represents whole numbers.

- With a 16-bit word length, the range of integer values typically spans from -32,768 to 32,767 (i.e., from (-2^{15}) to (2^{15} -1).

Signed Integer	Unsigned Integer
int	Unsigned int
short int	Unsigned short int
long int	Unsigned long int

Floating Data Type

- Floating-point types occupy 4 bytes (32 bits), with the last 6 bits used for precision.
  - The decimal point uses 1 bit.
- Types of floating-point data:
  - `float`: Occupies 4 bytes.
  - `double`: Typically occupies 8 bytes (64 bits).
  - `long double`: Often occupies 12 or 16 bytes (96 or 128 bits), depending on the system and compiler.

Character Data Type

- Character data type may be signed or unsigned.
- It is defined as `char` and occupies one byte (8 bits).

Bool Type

- A `bool` object can be assigned the literal values `true` or `false`.
  - Example: `bool found = false;`.
- `bool` cannot be declared as `signed`, `unsigned`, `short`, or `long`.

```
boolfound = false;
int occ count = O;
while( )
{
found = look_for();
occ_count +=found;
}
```

- `bool` objects and literals are implicitly promoted to `int` when an arithmetic value is required. `false` becomes `0`, and `true` becomes `1`.

- Arithmetic and pointer values are implicitly converted to `bool`.

  - A zero or null pointer value is converted to `false`.

  - All other values are converted to `true`.

Derived Data types

ARRAY TYPES

- An array is a collection of variables of the same type, all accessible via a common name.
  - Example: `int a[10];` declares an array of 10 integers.

- The dimension of an array must be a constant expression. You cannot use a non-constant variable to specify the size of an array.

- The dimension value must be known at compile time. Non-constant values are evaluated only at runtime, so they cannot be used to define array dimensions.

Example:

```
const int size = 5;
int a[size] = {0, 1, 2};
```

-An array can be explicitly initialized without specifying its dimension. If the number of elements provided is less than the specified dimension, the remaining elements are automatically set to zero.

- A character array can be initialized in two ways:
  - As a list of comma-separated characters: `const char ca1[] = {'C', '+', '+'};`
  - As a string literal: `const char ca2[] = "C++";`

For example: `const char ch3[3] = {'a', 'b', 'c'};`

Pointer type

- A pointer is a variable that stores the address of another variable.

- A pointer is defined by prefixing the identifier with the dereference operator (`*`): `int *p = 0; // declares a pointer to an integer`

- A pointer can hold a value of zero, indicating it points to no object. Pointers cannot hold non-address values.

- A `void*` pointer, also known as a generic pointer, can hold the address of any data type but cannot be dereferenced directly.

Example:

```cpp
void *gp;
int *ip;
gp = ip; // Assigns an integer pointer to a void pointer
// *gp = 10; // Illegal, as you cannot dereference a void pointer
```

Pointer Arithmetic:

- A number can be added to a pointer, which advances the pointer by that number of elements of its type.

- A number can be subtracted from a pointer, which moves the pointer back by that number of elements of its type.

- Incrementing a pointer advances it to point to the next element of its type.

String type

C++ provides two main representations for strings:

1. C-style Character String:
   - Represents strings as arrays of characters terminated by a null character (`'\0'`).
   - Example: `char str[] = "Hello";`

2. String Class Type:
   - Provides a higher-level, more flexible way to handle strings through the `std::string` class from the Standard Library.
   - Example: `std::string str = "Hello";`

String Class type

The `std::string` class in C++ supports the following operations:

a) Initialize a String:

- Initialize a string object with a sequence of characters or with another string object.
- Example: `std::string str1("Hello"); std::string str2 = str1;`

b) Copy a String:

- Copy the content of one string to another.
- Example: `std::string str2 = str1;`

c) Compare Strings:

- Compare two strings for equality or ordering.
- Example: `bool equal = (str1 == str2);`

d) Append Strings:

- Concatenate two strings.
- Example: `str1 += " World";`

e) Find String Length:

- Determine the length of the string.
- Example: `size_t length = str1.length();`

f) Check if a String is Empty:

- Determine if a string is empty.
- Example: `bool isEmpty = str1.empty();`

g) Define an Empty String:

- Initialize a string object with no content.
- Example: `std::string emptyStr;`

Reference Types

A reference provides an alias for an existing object. The general syntax is:

DataType &referenceName = variableName;

For example:

float total = 100;

float &sum = total;

In this case, both `total` and `sum` refer to the same data object in memory.

If you output their values with:

cout << total << "\n" << sum;

Both will display `100`.

If you modify the value:

total = total + 10;

Both `total` and `sum` will now have the value `110`.

You cannot initialize a reference with the address of a variable, as in:

int &rerefval = &ival; // illegal

While a reference behaves similarly to a pointer, it cannot be initialized with an object's address. Unlike a pointer, a reference must be initialized when declared and cannot be reassigned to refer to another object. Essentially, a reference is treated internally as a constant pointer.

The primary application of reference variables is in passing arguments to functions. For example:

```
void fun(int &x) {
   x = x + 10;
}
int main() {
   int m = 10;
   fun(m); // function call
   return 0;
}
```

When the function `fun` is called, the reference variable `x` is initialized as:

```
int &x = m;
```

This means `x` is an alias for `m`. Any modifications made to `x` inside the function `fun` will directly affect the value of `m`. So, after the function call, `m` will have the value `20`.

User-defined Data types

- STRUCTURES
- UNION
- CLASSES
- ENUMERATED DATA TYPE

Enumerated Data type

- It is a user-defined data type.
- The syntax of enum statement is similar to that of struct
- For Ex: enum shape{ circle, square, triangle};
- The tag name becomes new type name and new variables can be declared using these tag names.
- For Ex: shape ellipse;
- Here, ellipse is a variable of type shape.

- In C++, an integer value cannot be automatically converted to an enum value, but an enum value can be used in place of an integer.

- By default, enumerators are assigned integer values starting from 0, 1, and so on.

- Enumerators can also be explicitly assigned integer values. For example:

`enum color { red, blue = 4, green }`.

- Anonymous enums are enums without tag names. For example:

`enum { off, on };

int switch = off;`.

- Enumerations are useful for defining symbolic constants in switch statements.

Literals- constant Qualifier

- Literals are constant values assigned symbolic names to enhance readability and simplify the handling of standard constant values in C++.

- C++ provides three methods for defining constants:

1. `#define` preprocessor directive
2. Enumerated data types
3. `const` keyword

   For example: `const float PI = 3.1452;`

- The statement `const float PI = 3.1452;` declares a variable `PI` and assigns it the constant value `3.1452`.

- Without the `const` keyword, the variable `PI` could be modified, which is not desired for constants.

- In C++, using the `const` keyword ensures that the value of `PI` cannot be changed after its initial assignment, preventing accidental modification.

- The `const` qualifier is particularly useful for ensuring that constant values remain unchanged throughout the program, whether or not functions are involved.

Tokens, expressions and control structures

- Since C++ is a superset of C, most C constructs are valid in C++.

- Tokens: These are the smallest individual units in a program.

- In C++, tokens include keywords, identifiers, constants, strings, and operators.

- C++ has 48 keywords, 32 of which are inherited from C, with 15 more added by ANSI C++.

- Identifiers: These are names assigned to variables, functions, arrays, classes, etc., and follow the same rules as in C.

- A key difference between C and C++ is in the length of identifiers: ANSI C only recognizes the first 32 characters of a name, while C++ has no limit on identifier length, making all characters significant.

- Like C, C++ supports various types of literal constants.
- Examples include:
  - 123: Decimal constant
  - 3.14: Floating point constant
  - 075: Octal integer
  - 0x1A: Hexadecimal integer
  - "Hello": String constant
  - 'A': Character constant
- C++ also recognizes all the backslash escape characters from C, such as `\n`, `\t`, and `\\`.

Character constants

- Single character constants: These are individual characters enclosed within single quotes.

  Example: `'A'`, `'9'`, `'#'`

- String constants: These are sequences of characters enclosed within double quotes.

  Example: `"class"`, `"123"`, `"Hello, World!"`