## Introduction

*Background:*

In this book, the main aim is to write the application with easy understandable, easy maintainable, portability, scalability, high efficiency, and do any expected behaviors.

*The main strategies can be divided into two types: General Design Strategy and Specific Language Feature with Specific Details.*

The former strategy is mainly focused on discussing Design which is mainly focused on “How to choose the Design to finish among two different behaviors”.

Such as *“Choose Inheritance or Templates”*, *“Public Inheritance or Private Inheritance”*, *“Private Inheritance or Composition”*, *“Member function or Non - Member function”*, *“Pass - By - Value or Pass - By - Reference”*.

It’s important to make the right decision on these points, since one bad decision may not bring any bad result, but would show some bad results, then at that time to correct, it would be difficult and time - consuming, and the cost would be very high.

Even you totally know what to do, then it may still be difficult to go to the right norm. *What is the right return type of assignment Operator ? When should make Destruction Function Virtual ? When operator new can not find enough main memory, then what should be the next step ?*

Software Design and Realization is complex, which has been constrained by Hardware, Operating System, Constraint Condition of Application.

*Terminology:*

In this Chapter, would introduce “Terminology” that each Programmer should get knowledge with.

1. *Declaration* - So - Called Declaration is to tell *name and type of something to Compiler but to neglect all Details.*

Below are all Declarations:

* *extern int x;* - Declare one Integer Variable x.
* *std::size\_t numDigits(int number);* - Declare function definition with parameter number and return type of size\_t.
* std is the name space where almost all C++ Elements stay.
* size\_t is used in C++ which can be used in calculation number, and it is the non - negative integer, which can be considered as unsigned type.
* *std::size\_t numDigits(int number)* is called *Signature*, which includes parameter and return type. *Signature equals to type of function. std::size\_t (int number) is the signature of numDigits.*
* *std::size\_t is accepted parameter variable in vector, deque, and operator[ ] of string.*
* *class Widget;* - Declare class object Widget.
* *template<typename T>;*

*Class GraphNode;* - Declare template class definition GraphNode with typename T.

1. *Definition* - The main Task of Definition is to *provide more details to Compiler*.

* *For Objects - Definition is where Compiler provides Main Memory for it.*
* *int x;*
* *For Function or Function Template - Definition is where all code are provided.*
* *std::size\_t numDigits ( int number )*

*{*

*return number;*

*}*

* *For Class or Class Template - Definition is where all memberships provided.*
* *class Widget {*

*public:*

*Widget();*

*~ Widget();*

*};*

* *template <typename T>*

*Class GraphNode*

*{*

*public:*

*GraphNode();*

*~ GraphNode();*

*};*

1. *Initialization -*  The main Task is to *give initialized value to Parameter*.

* *User - Defined object, initialization should be executed by Constructor.*
* *Default Constructor is one function that is callable but with no parameter. Such constructor would has no parameter or has default value for each parameter.*
* *class A {*

*public:*

*A ( ); // Default Constructor.*

*explicit A ( int a = o, int b = 10 ); // Default Constructor.*

*explicit A ( int x ); // Default Constructor.*

*}*

*Attention that, here explicit is used to prevent implicit type conversion from happening but still supports the explicit type conversion. Normally, Explicit Default Constructor is much more popular than Non - Explicit Default Constructor, since it forbids the Compiler executing non - Excepted Type Conversion. Except there has a good reason to allow Default Constructor is being used as Implicit, otherwise, it is recommended to define it as Explicit.*

1. *Copy Constructor*

* *copy constructor is used as “Initialize itself by using the same type object”.*
* *copy assignment operator is used as “Initialize self by using the value of the same type object.”*

*class Widget()*

*{*

*public:*

*Widget ( ); // Default Constructor.*

*Widget ( const Widget & rhs ); // Copy Constructor.*

*Widget& operator = ( const Widget & rhs ); // Copy Assignment Operator.*

*}*

*Widget w1; // Calling Default Constructor.*

*Widget w2( w ); // Calling Copy Constructor.*

*w1 = w2; // Calling Copy Assignment Operator.*

* *Difference between Copy Constructor and Copy Assignment Operator:*
* If new object is being defined, then Copy Constructor would be called; It is impossible to call Copy Assignment Operator.
* If no new object is being defined, then Copy Assignment Operator is being called.

Copy Constructor is important since it helps define how the parameter would be passed over to another parameter.

*Pass - by - Value always means that “calling Copy Constructor Function”. Normally, it is a bad idea to pass user - defined parameter, Pass - By - Reference - To - Const is a much greater idea.*

1. *Standard Template Library* - STL is the so - called Standard Template Library and it is one part of the Standard Template Library, it contributes on vector, list, set, and map, iterator, algorithm, and related functions.
2. *Undefined Behavior* - Java/C# users may feel surprised about Undefined Behavior. Because of the some factors, some C++ constitutes may have no definition behavior, you can not estimate what’s happen during run time period.

* Int\* p = 0; cout << \*p; *( The pointer is the null pointer. )*
* char name [ ] = “Darla”; cout << name [ 10 ]; *( This has exceeded the legal range. There has only 6 places for name array, while the afterwards statement needs to print name [ 10 ]. )*

*The undefined behavior may cause wrong behavior, sometimes crash, sometimes can not generate the true answers.*

1. *Interface -* Java and .Net all provide Interfaces, however there has no such element in C ++. In C++, interface means the function signature or accessible element of class or general design.
2. *Client -* Client can be someone or something, he or it can use your code (normally your interface.) The client of function means users or caller or those write and maintain the code.

*When we mention Constructor or Destructor, sometimes I use ctor and dtor as abbreviation.*

*Naming Conventions:*

* *Parameter Naming Principle:* lhs => left - hand side and rhs => right - hand side.
* *Pointer Naming Principle:* pt => pointer to T
* Widget\* pw => pointer to Widget
* class Airplane;

Airplane\* pa; => pointer to Airplane

* class GameCharacter;

GameCharacter \* pgc; => pointer to GameCharacter

* *Reference Naming Principle:* ra => reference to A
* rw => reference to Widget
* ra => reference to Airplane

*Threading Consideration:*

*TR1 and Boost:*