2. Pointer and Reference in C++ and Reference in Java

2.1 Pointer(指標) in C++

說明:

1. 對於某資料型態 T，T* 之資料型態為「指到 T 型態資料之指標」(pointer to T)，T 可以是 primitive data type，如 int 或 double，或是 struct 或 class 等自定資料型態。

2. 指標如變數亦是代表一儲存資料之空間，不同的是它所能存放的內容是「記憶體位址」，某一資料型態之「變數」或「陣列元素」在記憶體中的位址(記憶體資料巢編號)，而非程式中欲直接運算之資料。

3. 指標宣告時指定了其所指資料之型態(T)，故釐定了所指資料是位於自指標所存位址起特定大小(depend on T)之記憶體空間，以及如何解讀此空間中之 binary data。

4. 使用指標，可依址取值(dereferencing)，存取其所指的記憶體空間內的資料，為除了使用變數或陣列元素外，另一種操作資料的方式。

5. 與變數或陣列不同的是，指標可經由指定或運算，改變其所指(對應)之記憶體資料巢(變數或陣列一旦宣告產生，於其生命週期內，其所對應之記憶體區塊為固定不變)，故可由單一指標所指位址之不斷移動，改變一群資料，如一陣列。

6. 一指標占記憶體大小為 2Bytes (for all type T)。

2.1.1 Pointer declaration

格式:

```
資料型態 * 指標名稱;
T*   Tptr;
```

例: Multiple declarations in a single line:

同時宣告一整數變數(a)與一整數指標變數(b)

```
int a, *b;          或      int* b, a;
```

宣告二浮點數指標變數:

```
double *x, *y;      或      double* x;    double* y;
```

2.1.2 Pointer Operations

1. **取址運算**：將指標「指到(point to)」某一變數 – 指標所存資料為位址，某變數的位址可由定址運算子(&)得到，之後將此記憶體位址值以指定運算子(=)指定(copy)給指標。變數之型態，需與指標宣告時指定之所指型態相同。

   ```
   指標 = &變數;
   ```

2. **依址取值**(dereferencing)：使用取值運算子(*)，可存取指標所指記憶體中的資料，對此記憶體中的資料做指定，讀取，或運算。.

   ```
   *指標 = value;
   cout << *指標;
   (*指標)++;
   ```

3. **算數運算**：指標可以遞增(++)或遞減(--)，整數(n)也可與指標相加(+或+=)或相減(-或-=)，這些運算的作用是改變指標所指之記憶體位址，於記憶體中前後移動，移動的單位是一個(++/--)或 n 個指標所指之資料型態(T)所佔記憶體空間大小，這些運算通常用於操作一陣列。

   ```
   指標++;
   指標--;
   指標+= n;
   指標-= n;
   ```

例:

```
int y = 5;
int *yPtr;
yPtr = &y;
cout<<"*yPtr ="<< *yPtr <<endl;
```

程式範例: oop_ex13.cpp oop_ex14.cpp

2.1.3 const with pointer

1. 如 const 設於變數型態(T)之前，則此指標為唯讀指標，無法使用其來更改其所指記憶體空間內之儲存值，但指標所指位址可改變。欲指到常數的位址，需使用唯讀指標。

   ```
   const   T*   T_ptr;
   ```

2. 如 const 設於*之後，則此指標為定址指標，其所指的位址固定不可改變，但其所指之記憶體空間內之儲存值可改變。

   ```
   T*   const   T_ptr;
   ```

程式範例: oop_ex15.cpp

2.1.4 void 指標

1. void 指標可指向任何型態的變數。

2. void 指標只具有某資料的起始位址，而無此資料的型態(佔記憶體大小，如何解讀，故存取此資料(依址取值)時常需以強制轉換方式指定其型別。

程式範例: oop_ex16.cpp

2.2. reference(參考) in C++

說明:

1. C++除了 primitive data types 及指標外，另提供了參考型態，參考可視為某變數的「別名」。其之宣告為在資料型態後加上參考運算子(&)，所產生的變數即為一參考，且宣告時必須要同時給予初值(其所參考到的變數)。

2. 參考是其所參考到變數的別名，它們對應到相同的一塊記憶體資料，對某參考的值做改變即是對其所參考到之變數的值做改變，反之亦然。

3. 一參考變數所占記憶體大小及其運算方法與其參考到的變數相同。

4. 參考的主要作用，在於函式(function)之參數傳遞。

格式:

```
資料型態&    參考變數  =  變數;
T&    T_ref = variable_name;
```

例: 宣告一整數變數 a 之參考

```
int a;
int& a_ref = a;
```

程式範例: oop_ex17.cpp

2.3. reference(參考) in Java

1. Java 對於 primitive data types，只能產生變數，並無指標或參考，也無取址(&)之運算。

2. Java 對於陣列或物件等型態之資料，無法產生直接代表這些資料的變數，而是透過稱為參考(reference)型態之變數來操作，Java 之參考有如 C++之指標，其儲存值為陣列或物件的記憶體位址(memory address)。

3. Java 之參考並不像 C++之指標或參考需使用 * 或 & 明確地宣告，而是根據所產生的資料類型而定，例如宣告 T var_name，如 T 為 primitive type，則 var_name 為變數，如 T 為類別或陣列，則 var_name 為參考。

4. Java 之參考不支援依址取值(deferencing)之運算，以及++或--等之算數運算，只可透過參考用 [ ] 運算子存取其所指陣列之元素，或用「 . 」運算子存取其所指物件之(public)成員。

5. 對某參考做指定運算(=)，並非改變其所指的物件，而是改變此參考所指的位址，至等號右邊的位址。例如: ref1 = ref2，則 ref1 與 ref2 均指到 ref2 所指之物件。

例:

```
class test{
public:
    int a;
    double b;
    test(){a=0; b=0;}
    ……
}
```

```
int data1 = 10;                 // variable
int [ ] data2 = {1, 3, 5, 7, 9};    //reference
test data3 = new test();        //reference
test data4 = new test();        //reference

data2[2] = data1;
data3.a = data1;
data4 = data3;
```

程式範例: oop_ex_a.java