

# Homework 2 Part 1

---

## Dynamic array tutor2.cpp

班級	學號	姓名	日期
四機械四乙	B10831020	吳宇昕	10/1/2022

[source code](#) and [replit url](#)

Q1: 將函式變數由 `int*` 改為 `int[]`

兩種情況下，函式都能正常運作。迴圈最後一行使用的`++`運算子作用於`int pointer`時，每次都會自動偏移4個byte，也就是`int`的大小。因此無論函式的引述宣告為`[]`或`*`都可以順利執行。

Q2: 把`push_back`的for loop寫成函式

```
void pushStuffIntoVtr(vector<int>& dynArray, int count)
{
    for ( int i=0; i<count; ++i){
        int newValue = 1000+ i*10;
        dynArray.push_back(newValue);
    }
}
```

Vector以pass by reference方式傳入函式，方便修改其記憶體位置的值。同時，pass by reference可以避免複製整個vector至另一個函式的stack frame，節省記憶體使用量。

## 心得

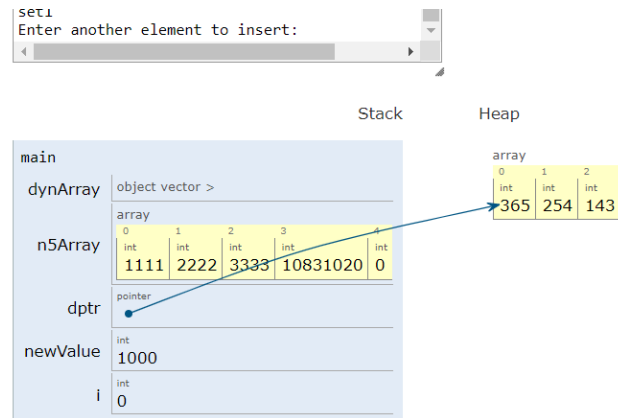
這項作業介紹了tutor這良好的工具，可以視覺化呈現各個變數的數值，以及pointer所指向的位置。甚至當pointer正指向一個無效的記憶體位置時，還會顯示屎的表情符號，清楚呈現一個失控的pointer可以摧毀人的一天。

其中有個問題讓我花了非常多時間理解，就是為何原程式執行37行以前，`dptr`是有效的，但是`dynArray`一旦被38行加入新值之後，馬上失效？下圖為`dynArray`被38行`push_back`之前情形，`dptr`仍是有效的pointer

```

24 vector<int> dynArray (3); // dynamic array of int
25 int n5Array[5] = {1111, 2222, 3333, 10831020}; // add
26 dynArray[0] = 365;
27 dynArray[1] = 254;
28 dynArray[2] = 143;
29
30 cout << "Number of integers in array: " << dynArray
31 // set1
32 int* dptr = &dynArray[0];
33 printPTR( dptr, "start 0", "set1", 3 );
34 cout << "Enter another element to insert: " << endl
35 int newValue = 10831020; // TODO: your stuID
36 for ( int i=0; i<5; ++i){
37     newValue = 1000+ i*10;
38     dynArray.push_back(newValue);
39 }
40 dptr = &dynArray[3];
41 cout << "pointered value = " << *dptr;
42
43 printPTR( dptr, "start 3", "New Added 5 elements?";

```

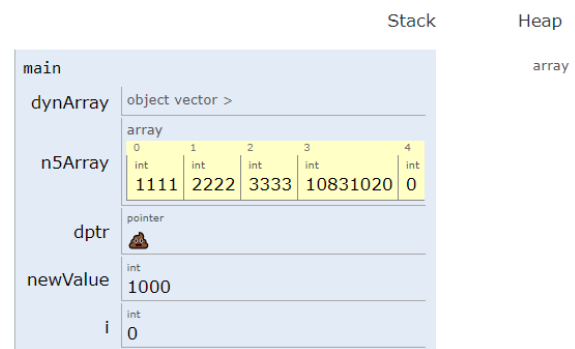
[Edit this code](#)


下圖為38行push\_back以後，dptr已經失效

```

27 dynArray[1] = 254;
28 dynArray[2] = 143;
29
30 cout << "Number of integers in array: " << dynArray
31 // set1
32 int* dptr = &dynArray[0];
33 printPTR( dptr, "start 0", "set1", 3 );
34 cout << "Enter another element to insert: " << endl
35 int newValue = 10831020; // TODO: your stuID
36 for ( int i=0; i<5; ++i){
37     newValue = 1000+ i*10;
38     dynArray.push_back(newValue);
39 }
40 dptr = &dynArray[3];
41 cout << "pointered value = " << *dptr;
42
43 printPTR( dptr, "start 3", "New Added 5 elements?";

```

[Edit this code](#)


在vscode仔細檢查前後dynArray記憶體位置後，才發現vector被push\_back之後，它的記憶體位置可能會截然不同。下圖顯示push\_back前dynArray的起始記憶體位置，是0x1011630

WATCH

- dptr: -var-create: unable to create vari...
- dynArray: {...}
- std::\_Vector\_base<int, std::allocator<int>...>
  - \_M\_impl
    - > std::allocator<int> (base): std::all...
    - > \_M\_start: 0x1011630
    - > \_M\_finish: 0x101163c
    - > \_M\_end\_of\_storage: 0x101163c

```

50 void pushStuffIntoVtr(vector<int>& dynArray, int count)
51 {
52     for ( int i=0; i<count; ++i){
53         int newValue = 1000+ i*10;
54         dynArray.push_back(newValue);
55         system("pause");
56     }
57 }

```

下圖顯示push back之後dynArray起始記憶體位置，是0x1017018，前後相距甚遠。

WATCH

- dptr: -var-create: unable to create vari...
- dynArray: {...}
- std::\_Vector\_base<int, std::allocator<int>...>
  - \_M\_impl
    - > std::allocator<int> (base): std::all...
    - > \_M\_start: 0x1017018
    - > \_M\_finish: 0x1017028
    - > \_M\_end\_of\_storage: 0x1017030

```

50 void pushStuffIntoVtr(vector<int>& dynArray, int count)
51 {
52     for ( int i=0; i<count; ++i){
53         int newValue = 1000+ i*10;
54         dynArray.push_back(newValue);
55         system("pause");
56     }
57 }

```

由於vector需要找更大的記憶體空間容納新進數值，它會在heap上找出一個比原vector大一點的記憶體空間，把舊有的數值複製過去，並將新數值放入新記憶體位置的最後一個空間。dptr在程式執行到dptr

`= &dynArray[3]`以前完全不知道vector的記憶體位置更新過，傻傻的指向舊的、已經過時被清除的`dynArray`的位置。因此被tutor標記為失控的pointer。

在電腦所有記憶體耗盡前，vector似乎可以無限增長，但是持續讓它配置新記憶體空間、複製舊有值、回傳新記憶體位置，恐怕會傷害程式效能。