TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN
MÔN: **KĨ THUẬT LẬP TRÌNH**

HƯỚNG DẪN THỰC HÀNH

TUẦN 01

BIẾN CON TRỎ VÀ KĨ THUẬT BỘ NHỚ ĐỘNG

TP.HCM, ngày 10 tháng 05 năm 2020

MỤC LỤC

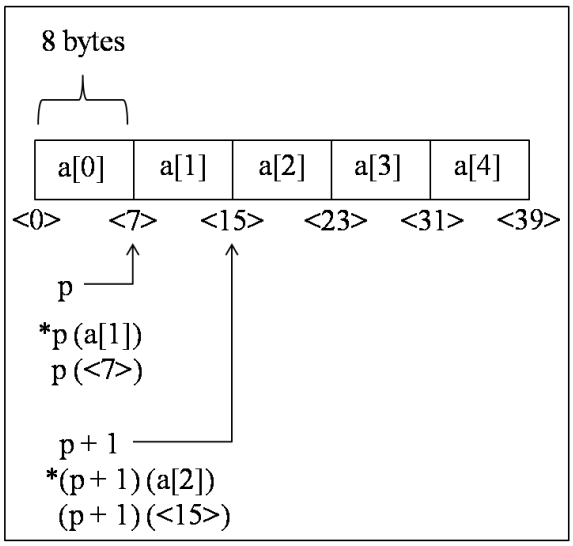
1 Khái niệm biến con trỏ:

Biến con trỏ là địa chỉ ô nhớ. Ví dụ int* p là một biến con trỏ có tên là p, giá trị của p là địa chỉ của một biến thông thường kiểu int.

Xét ví dụ sau:

```
double a[5];
double *p;
p = &a[1]
```

Do biến kiểu double cần 8 bytes lưu trữ nên ta có mảng gồm 5 phần tử, mỗi phần tử 8 bytes vậy tổng số cần là 40 bytes.



Ta thấy phép cộng với kiểu con trỏ có chút khác biệt. Ví dụ p đang trỏ tới địa chỉ <7> thì (p + 1) sẽ trỏ tới <15>, có nghĩa là p + 1 $\Rightarrow$ p + 1 × sizeof(double) $\Rightarrow$ <7> + 8 = <15>. Khi làm việc với con trỏ, ta cần biết các kí hiệu cơ bản sau: p chính là địa chỉ mà p đang trỏ tới, ví dụ trong trường hợp trên là <7>, trong khi đó *p là giá trị tại ô nhớ mà p đang trỏ tới, trong trường hợp này là a[1], và cuối cùng là &p, trong trường hợp này &p là địa chỉ của chính con trỏ p (Hình vẽ không có địa chỉ của con trỏ p). Nếu ta muốn truy xuất tới phần tử a[0], ta cần dòng mã *(p – 1).

2 Vùng nhớ Heap và Stack.

Ta lưu ý, khi dùng biến con trỏ ta thường xin cấp phát bộ nhớ. Ví dụ int* p = **new** int[5], tức là con trỏ p kiểu int xin cấp phát 5 phần tử kiểu int trong vùng nhớ Heap. Sauk hi xin cấp phát, ta sẽ sử dụng vào mục đích nào đó và khi kết thúc ta cần dòng mã **delete**[] p để

xóa các phần tử xin cấp phát. Nếu thiếu dòng mã này thì vùng nhớ Heap sẽ vẫn còn chứa 5 phần tử và hiện tượng này gọi là rò rỉ bộ nhớ.

Trường hợp ví dụ trong phần 1, ta thấy con trỏ p nhận địa chỉ từ một mảng khai báo tĩnh trong stack, vì vậy ta không cần dùng dòng mã **delete** để xóa các vùng nhớ trong stack.

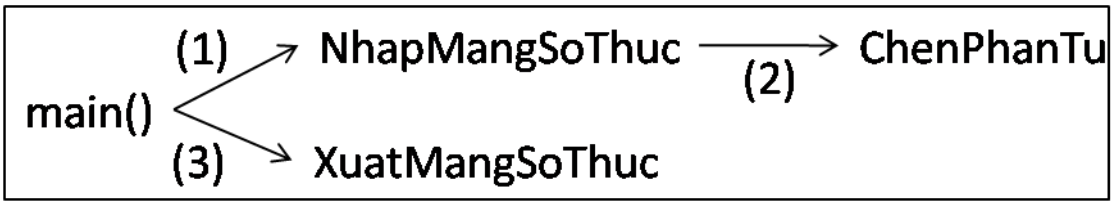
3 Ví dụ minh họa.

Trong phần này ta sẽ được giới thiệu qua các đoạn mã làm việc với biến con trỏ và mảng một chiều. Đoạn mã gồm các hàm nhập mảng, xuất mảng, và thêm một phần tử vào mảng.

Dòng	
0	#include <malloc.h>
1	#include <iostream>
2	using namespace std;
3	void XuatMangSoThuc(float a[], int n){
4	for(int i = 0; i < n; i++)
5	cout << a[i] << " ";
6	}
7	
8	void ChenPhanTu(float*& a, int& n, float x){
9	int m = n + 1;
10	float* anew = (float*)realloc(a, m * sizeof(float));
11	if(anew != NULL){
12	anew[n] = x; n++;
13	a = anew;
14	}
15	}
16	
17	void NhapMangSoThuc(float*& a, int &n){
18	float x;
19	a= NULL; n = 0;
20	while(cin >> x){
21	ChenPhanTu(a, n, x);
22	}
23	}
24	
25	void main(){
26	float* B; int nB;
27	cout << "Nhap cac phan tu so thuc vao mang: " << endl;

```
28    NhapMangSoThuc(B, nB);
29    cout << nB << " phan tu: " << endl;
30    XuatMangSoThuc(B, nB);
31    if(B != NULL){
32       delete[] B;
33    }
34 }
```

Ta sẽ xem xét lần lượt từng hàm. Hàm **XuatMangSoThuc** với đầu vào là mảng số thực và số lượng phần tử, hàm này không có đầu ra, chức năng chính là in ra màn hình các phần tử trong mảng. Hàm **ChenPhanTu** với đầu vào là mảng số thực, số lượng phần tử và phần tử cần chèn, ta lưu ý ký pháp phần tử đầu tiên là *&, ở đây ta dùng tham chiếu con trỏ, ngoài ra ta còn dùng thêm hàm realloc để thay đổi số phần tử của mảng. Ta khai báo thêm mảng **anew** lí do là vì hàm **realloc** rất có thể xóa mảng cũ **a** để tạo mảng mới, vì vậy để an toàn ta khai báo mảng **anew**, sau đó gán lại cho mảng **a**. Hàm cuối cùng là **NhapMangSoThuc**, đầu vào của hàm là mảng số thực và số phần tử, trong hàm này ta gọi lại hàm **ChenPhanTu** để lần lượt thêm vào mảng. Bên dưới là sơ đồ gọi hàm.



4 Con trỏ cấu trúc

Khái niệm cấu trúc được giới thiệu để mở rộng các kiểu dữ liệu cơ sở. Xét kiểu PHANSO ta có:

```
struct phanso{
  int tuso, mauso;
};
typedef struct phanso PHANSO;
```

Để thực hiện việc nhập xuất với kiểu PHANSO, ví dụ như cin >> ps hay cout << ps. Ta cần quá tải hai toán tử này với kiểu PHANSO.

```
ostream& operator<< (ostream& outDev, const PHANSO& ps){
  if(ps.mauso == 1 || ps.tuso == 0) outDev << ps.tuso;
  else outDev << ps.tuso << "/" << ps.mauso;
  return outDev;
}
istream& operator>> (istream& inDev, PHANSO& ps){
```

```
  inDev >> ps.tuso >> ps.mauso;
  return inDev;
}
void main()
{
  PHANSO* ps = new PHANSO;
  cin >> (*ps);
  cout << (*ps);
}
```

Ta thấy đầu vào của toán tử >> cần một thiết bị nhập (kiểu istream) và một biến tham chiếu kiểu PHANSO (đó là lí do ta dùng kí pháp *ps). Tương tự cho toán tử <<. Ngoài ra để truy xuất tới các phần tử trong kiểu PHANSO, ta dùng kí hiệu '->'. Ví dụ ps->tuso hoặc ps->mauso.

5 Bài tập.

Xem kĩ các ví dụ trên, sinh viên hãy xây dựng mảng PHANSO gồm các hàm có chức năng như sau:

- Nhập mảng các phân số

- Xuất mảng các phân số

- Chèn một phân số vào cuối mảng

- Xóa một phân số vào cuối mảng

- Hủy mảng phân số vừa xin cấp phát.

Sinh viên tự thiết kế đầu vào và đầu ra của tất cả các hàm, sau đó viết hàm main() minh họa các hàm vừa xây dựng. Xem ví dụ gợi ý hàm main bên dưới

```
void main()
{
  PHANSO* p;
  int n;
  NhapMangPhanSo(…);
  cout << "Mang phan so sau khi nhap: " << endl;
  XuatMangPhanSo(…);
  PHANSO* a = new PHANSO{2,3};
  ChenPhanTuPhanSo(…, a);
  cout << "Mang phan so sau khi chen (2/3): " << endl;
  XuatMangPhanSo(…);
```

```
   XoaPhanTu(…, a);
   cout << “Mang phan so sau khi xoa (2/3): ” << endl;
   XuatMangPhanSo(…);
   //…các hàm hủy vùng nhớ vừa xin cấp phát
}
```