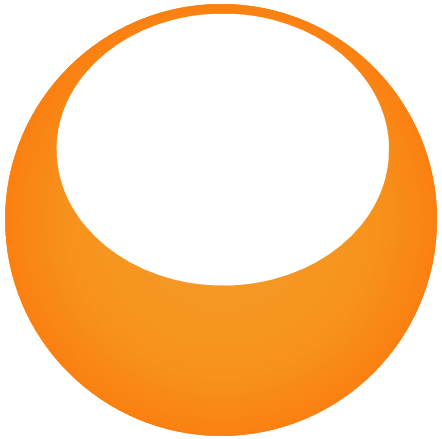
Phương pháp lập trình hướng đối tượng

Tuần 04:

Toán tử, 3 vấn đề con trỏ

Phạm Tú San
ptsan@fit.hcmus.edu.vn



TOÁN TỬ

Hàm toán tử

- Toán tử à một loại phương thức đặc biệt của lớp

```
PhanSo a(3,2), b(4,5), c;
c = a.Cong(b);
c = a + b;
```

- Trong C++, dùng từ khóa operator.

```
PhanSo operator +(const PhanSo &p1, const PhanSo &p2);
PhanSo p1, p2;
PhanSo p3 = p1 + p2;
```

- Có thể nạp chồng hàm cho toán tử

```
PhanSo operator +(const PhanSo &p, int iNumber);
float opeartor +(const PhanSo &p, float iNumber);
```

Hàm toán tử

- Phân loại hàm toán tử:
  - Toán tử độc lập:
    - Không thuộc lớp nào.
    - Ngôi của toán tử là số tham số truyền vào.

      PhanSo operator +(***const PhanSo &p1, const PhanSo &p2***);

      bool operator >(***const PhanSo &p1, const PhanSo &p2***);
  - Toán tử thuộc lớp:
    - Là phương thức của lớp.
    - Ngôi của toán tử: đối tượng của lớp + số tham số.

      PhanSo PhanSo::operator +(***const PhanSo &p***);

      bool PhanSo::operator >(***const PhanSo &p***);
  - **Cách sử dụng 2 loại là như nhau!!**

Các toán tử có thể viết chồng

+	-	*	/	%	^	&
|	~	!	=	<	>	+=
-=	*=	/=	%=	^=	&=	|=
<<	>>	>>=	<<=	==	!=	<=
>=	&&	||	++	--	->*	,
->	[]	()	new	new[]	delete	delete[]

- Các toán tử :: hay . hay .* không được phép định nghĩa bởi người dùng
- Toán tử: sizeof, typeid, ?: không được định nghĩa chồng
- Toán tử =, ->, [], () chỉ được viết chồng bằng các hàm non-static

Cú pháp chung

<kiểu trả về> **operator** <toán tử> (danh sánh tham số)

Ví dụ:

```
bool HoTen::operator==(const HoTen& rhs)
{
    return((sTen==rhs.sTen) && (sHo==rhs.sHo));
}
```

Cách sử dụng

```
int main()
{
    SinhVien sv1, sv2;
    if (sv1 == sv2) //sv1.operator==(sv2)
    {
       ...
    }
    ...
}
```

Một số lưu ý khi viết chồng toán tử

- Tránh thay đổi ý nghĩa nguyên thủy của toán tử đó
- Các cặp toán tử có cùng chức năng, ví dụ x=x+y và x+=y phải được viết cùng nhau và có cùng chức năng.
- Nếu toán tử chồng không là hàm thành viên của lớp thì nên sử dụng từ khóa **friend** thay vì truy xuất đến các thành phần dữ liệu 1 cách phức tạp

Toán tử không thuộc lớp

- Ví dụ: toán tử nhập xuất

```
friend ostream& operator<<(ostream &out, const PhanSo &src);
```

```
ostream& operator<<(ostream &out, const PhanSo &src)
{
        out<<src.tu<<"//"<<src.mau;
}
```

Toán tử gán bằng

- Tương tự như hàm dựng sao chép, nếu mỗi lớp đối tượng không có toán tử gán bằng thì trình biên dịch sẽ tạo 1 hàm toán tử gán bằng mặc định
- Hàm này cũng có chức năng tương tự như hàm dựng sao chép mặc định: sao chép từng bit của đối tượng nguồn cho đối tượng đích.

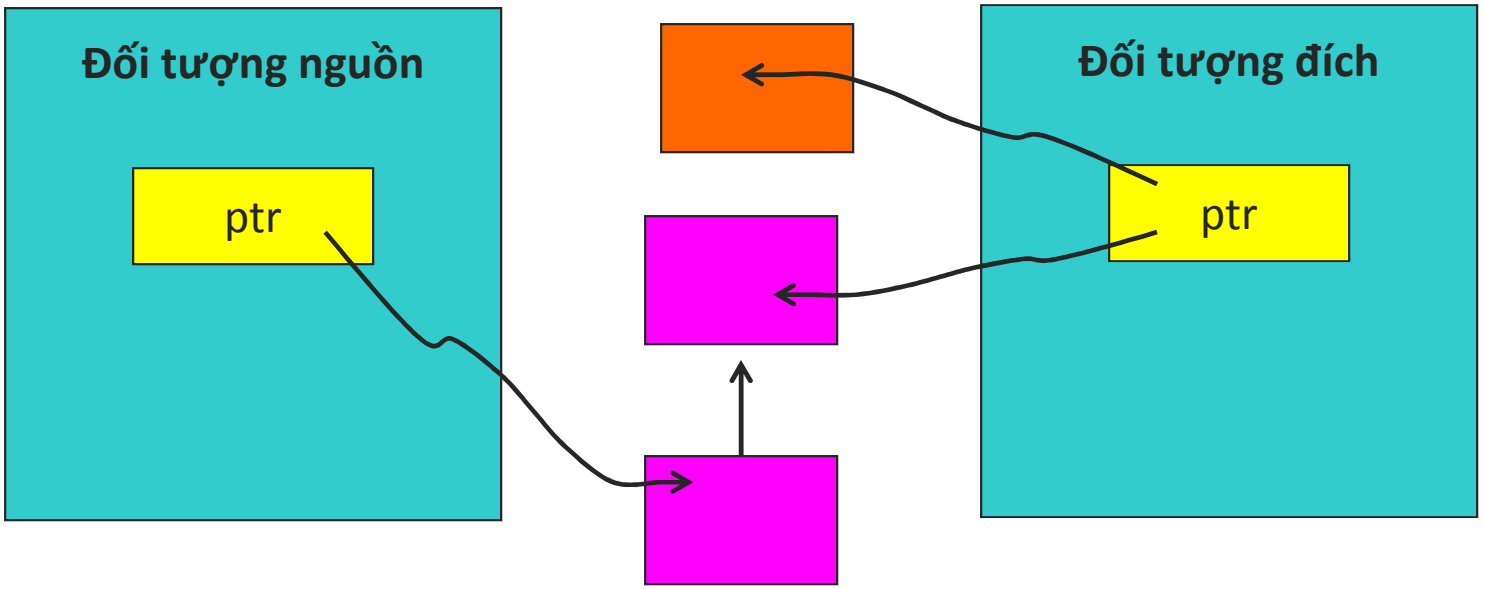
Toán tử gán bằng

- Do vậy, nếu lớp đối tượng có biến con trỏ và có nhu cầu gán bằng 1 đối tượng khác.

  ➔ cần xây dựng toán tử gán bằng cho lớp

- Lưu ý: toán tử gán bằng khác hàm dựng sao chép 1 số điểm sau:
  - Xóa phần bộ nhớ nó đang kiểm soát trước khi gán bằng đối tượng mới.
  - Kiểm tra kỹ việc đối tượng tự gán bằng chính nó.

Toán tử gán bằng



- Xóa bỏ vùng nhớ nó đang kiểm soát
- Copy vùng nhớ và trỏ quả vùng nhớ mới

Ví dụ - toán tử gán bằng

```
HocSinh& HocSinh::operator=(const HocSinh& h)
{
    if (this != &h)
    {
       delete [] this->HoTen;
       int size = h.HoTen.length();
       this->HoTen = new char[size];
       strcpy(this->HoTen, h.HoTen);
    }
    return *this;
}
```

3 vấn đề con trỏ

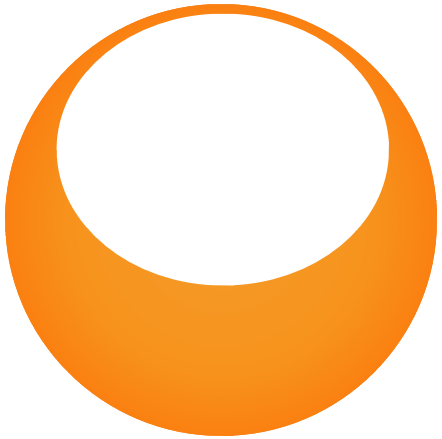
Bộ 3 hàm sau luôn đi chung với nhau:

- Hàm dựng sao chép (copy constructor)
- Toán tử gán bằng (assignment operator)
- Hàm hủy (destructor)

Hàm dựng sao chép – hàm hủy

```
HocSinh::HocSinh(const HocSinh& h)
{
	if(h.HoTen != null)
	{
		int size = h.HoTen.length();
		this->HoTen = new char[size];
		strcpy(this->HoTen, h.HoTen);
	}
}
```

```
HocSinh::~HocSinh()
{	if(this->HoTen != null)
	{
		delete []this->HoTen;
		this->HoTen = null;
	}
}
```



BÀI TẬP

Bài tập – 4.1

- Xây dựng các toán tử sau cho lớp Phân Số
  - Toán tử `+, -, *, /` , >, <, >=, <=,
  - Toán tử ++, -- theo dạng tiền tố và hậu tố (++x và x++)
  - Toán tử ép kiểu (int), (float)
  - Toán tử nhập, xuất
- Giả sử ta có phân số: `PhanSo x`

  Cần xử lý một số vấn đề sau:
  - Tính: `x + 5`
  - Tính giao hoán của phép cộng: `5 + x`
  - `Thảo luận để tìm ra giải pháp?`

Bài tập – 4.2

- Đối tượng “mảng số nguyên” có 2 thuộc tính là mảng động chứa các đối tượng Số Nguyên và số phần tử hiện có trong mảng.
- Xây dựng các phương thức giải quyết 3 vấn đề về con trỏ cho mảng

```
class MangSoNguyen
{
      private:
            int *GiaTri;
            int SoPhanTu;
};
```

Tham khảo

- Slide PPLTHĐT của
  - Thầy Nguyễn Minh Huy
  - Thầy Đinh Bá Tiến

Phương pháp lập trình hướng đối tượng

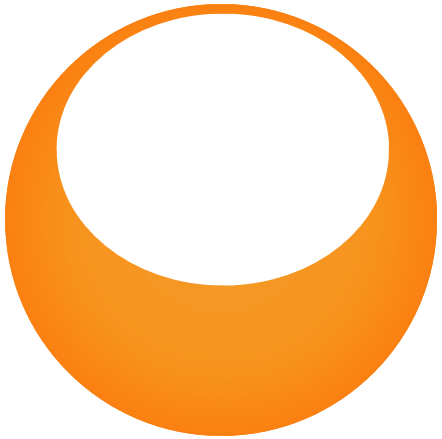
Tuần 04:

Một số vấn đề khác

Phạm Tú San
ptsan@fit.hcmus.edu.vn

Nội dung

- const
- Thành phần tĩnh
- Template
- Thư viện C++



CONST

Từ khóa const

- Từ khóa const đổi một biến, đối tượng thành hằng (nghĩa là biến đó chỉ được phép đọc.)
  - Biến, đối tượng được coi là hằng thì không được cập nhật các thành phần dữ liệu của nó.
  - Hàm thành phần được gọi là hằng thì không được sửa đổi thành phần dữ liệu của đối tượng gọi hàm

Ví dụ

- Một thành viên dữ liệu của lớp có thể được định nghĩa như hằng

Ví dụ:

```
class CPoint
{
        private:
                const float mMaxX;
                const float mMaxY;
        //...
};
```

Khởi tạo biến có từ khóa const

```
class CPoint
{
   private:
                const float mMaxX = 1024.0;
                const float mMaxY = 768.0;
   //...
};
// Cách khởi tạo trên là sai
```

Khởi tạo biến có từ khóa const (tt)

Trong hàm constructor

```
CPoint::CPoint(float x, float y): mMaxX(1024), mMaxY(768)
{
    …
}
```

Hàm thành viên là const

- Các hàm thành viên cũng có thể được định nghĩa như là hằng. Điều này được sử dụng để đặc tả các hàm thành viên nào của lớp có thể được triệu gọi cho một đối tượng hằng.

Ví dụ:

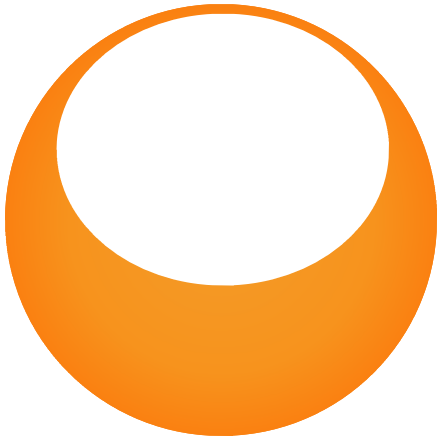
```
int docNgay() const; // hàm readonly
```

Hàm thành viên hằng

- Một đối tượng hằng chỉ có thể được sửa đổi bởi các hàm thành viên hằng của 1 lớp

```
class CSet
{
      public:
                        CSet(void){ card = 0; }
                        bool member(const int) const;
                        void addElem(const int);
                        //...
};
```

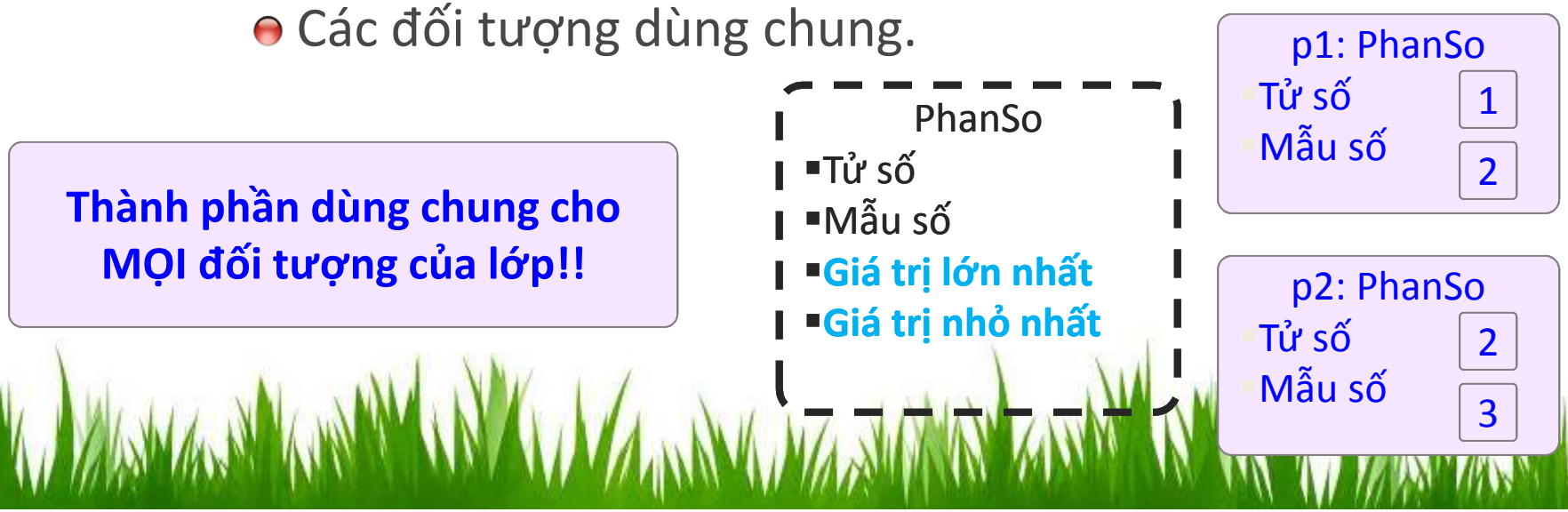
```
const CSet s;
s.addElem(10); // sai!!! addElem không là thành viên hằng
s.member(10); // ok
```



THÀNH PHẦN TĨNH

Thành phần tĩnh

- Thành phần của lớp (class members):
  - Thành phần đối tượng (instance members).
    - Thuộc tính và phương thức thông thường.
    - Mỗi đối tượng có bản sao riêng.
  - Thành phần tĩnh (static members).
    - Thuộc tính và phương thức tĩnh.
    - Các đối tượng dùng chung.



Thành phần tĩnh

- Khai báo và sử dụng:
  - Dùng từ khóa static.
  - Truy xuất bằng toán tử ::
  - Tham khảo trang 119 giáo trình HĐT

```
class PhanSo
{
private:
        static int GiaTriLN;
public:
        static int layGiaTriLN();
private:
        int         Tu;
        int         Mau;
};
```

```
PhanSo::GiaTriLN = 10000;

void main()
{
    PhanSo   p1(1, 2);
    PhanSo   p2(2, 3);

    int x1 = PhanSo::layGiaTriLN();
    int x2 = p1.layGiaTriLN();
}
```

Thành phần dữ liệu có kiểu là lớp

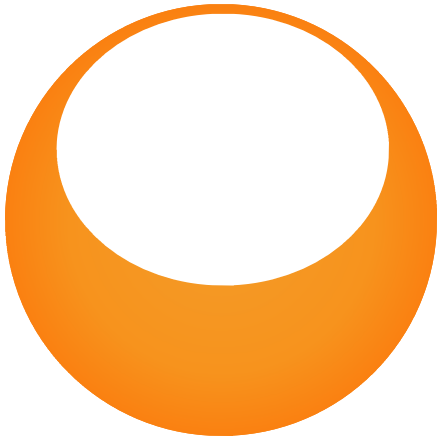
- Lớp B có thành phần dữ liệu là lớp A
  - Thứ tự khởi tạo và hủy?
  - Khởi tạo thành phần dữ liệu?

```
class Diem
{
private:
        int mX;
        int mY;
}
class DuongTron
{
private:
        Diem mTam;
        float mBanKinh;
}
```

Khởi tạo bằng danh sách khởi tạo

TênLớp (DSTSố): TênBiến1 (ThamSố), TênBiến2 (ThamSố)

```
class DuongTron
{
private:
        const float mBKToiDa;
        float mBanKinh;
        Diem mTam;
public:
        DuongTron():mBKToiDa(10){}
        DuongTron(float bktd, float bk, float x, float y): mBKToiDa(bktd), mTam(x,y)
        {
                mBanKinh = bk;
        }
        DuongTron(float bk, Diem a): mTam(a)
        {
                mBanKinh = bk;
        }
}
```



TEMPLATE

Function Template

- Xét hàm tìm min giữa 2 số:

```
int timMin(int a, int b)
{
        return (a < b) ? a : b;
}

float timMin(float a, float b)
{
        return (a < b) ? a : b;
}

PhanSo timMin(PhanSo a, PhanSo b)
{
        return (a < b) ? a : b;
}
```

Dùng Function Template!!

Function Template

- Hàm tìm min dùng Function Template:

```
template <class T>
T timMin(T a, T b)
{
    return (a < b) ? a : b;
}
```

Phải xây dựng toán tử < nếu kiểu dữ liệu T không phải là kiểu dữ liệu được dựng sẵn

```
void main()
{
    int a = 5, b = 3;
    int c = timMin(a, b);

    float d = timMin(1.5, 2.3);

    PhanSo p1(1, 2);
    PhanSo p2(1, 3);
    PhanSo p3 = timMin(p1, p2);
}
```

Function Template

- Đặc điểm của Function Template:
  - Hàm tổng quát cho nhiều kiểu dữ liệu khác nhau.
  - Tham số hóa kiểu dữ liệu.
  - Kiểu cụ thể được quyết định khi gọi hàm.
- Ghi chú:
  - Từ khóa **"class"** có thể thay bằng **"typename"**.
  - Phần khai báo và cài đặt đều có khai báo template.
  - Phần cài đặt hàm phải nằm cùng file:
    - Phần khai báo hàm.
    - Phần gọi sử dụng hàm.

Class Template

- Xét lớp đối tượng Mang:

```
class MangNguyen
{ private:
        int         miKichThuoc;
        int         *mpDuLieu;
        public:
        Mang(int iKichThuoc);
        int LayPhanTu(intiViTri);
};
        class MangPhanSo
{ private:
        int         miKichThuoc;
        PhanSo  *mpDuLieu;
        public:
        Mang(int iKichThuoc);
        PhanSo LayPhanTu(int iViTri);
};
```

Dùng Class Template!!

Class Template

- Lớp Mang dùng Class Template:

```
template <class T>
class Mang
{
private:
        int         m_iKichThuoc;
        T           *m_pDuLieu;
public:
        Mang(int iKichThuoc);
        T layPhanTu(int iViTri);
};
```

```
void main()
{
        Mang<int>           m1(10);
        int   a = m1.layPhanTu(5);

        Mang<PhanSo> m2(5);
        PhanSo   p = m2.layPhanTu(2);
}
```

Class Template

- Đặc điểm của Class Template:
  - Lớp tổng quát cho nhiều kiểu dữ liệu khác nhau.
  - Tham số hóa kiểu dữ liệu.
  - Kiểu cụ thể được truyền vào khi tạo đối tượng.
- Ghi chú:
  - Từ khóa **"class"** có thể thay bằng **"typename"**.
  - Phần cài đặt lớp phải nằm cùng file:
    - Phần khai báo lớp.
    - Phần tạo và sử dụng đối tượng của lớp.

  ➔ **Viết cài đặt bên trong lớp khi dùng Template.**

Nội dung

- Thành phần tĩnh
- const
- Template
- **Thư viện C++**

Thư viện C++

- Khái niệm thư viện:
  - Tập hợp những lớp, hàm có sẵn giúp giải quyết công việc thường gặp.
  - Bộ công cụ hữu ích của lập trình viên.
  - Một vài thư viện C++:
    - Thư viện chuẩn (C++ Standard Library).
    - Thư viện boost.
      - http://www.boost.org/
    - Thư viện MFC (**M**icrosoft **F**oundation **C**lasses).
      - Visual C++ - Windows

Thư viện C++

- Thư viện chuẩn:
  - Thư viện cơ bản nhất của C++.
  - Các lớp và hàm nằm trong namespace std.
  - File Header không .h.
  - Phân nhóm:
    - Nhóm nhập xuất: iostream, iomanip, fstream, …
    - **Nhóm STL.**
    - …
    - Thư viện chuẩn C: file header **c**xxx.

Thư viện C++

- Thư viện STL (**S**tandard **T**emplate **L**ibrary):
  - Một phần của thư viện chuẩn.
  - Các lớp và hàm hỗ trợ lập trình với template.
  - Phân nhóm:
    - Nhóm container: vector, list, deque, set, …
    - Nhóm string: string, …
    - Nhóm iterator.
    - …
  - Tham khảo trang 67 giáo trình HĐT

Thư viện C++

- Lớp string:
  - File header <string>.
  - Lớp đại diện cho các đối tượng chuỗi.
  - **Giải quyết 3 vấn đề con trỏ.**
  - Các phương thức chính:
    - string(char *): khởi tạo từ một chuỗi ký tự.
    - length(): lấy chiều dài chuỗi.
    - Toán tử [ ]: lấy ký tự tại một vị trí nào đó.
    - Toán tử >, <, ==, >=, <=, !=: so sánh theo thứ tự từ điển.
    - Toán tử +, +=: nối chuỗi.
    - find(char *): tìm chuỗi con.
    - substr(int, int): lấy chuỗi con.

Thư viện C++

- Ví dụ lớp string:

```
void main()
{
        string      s1(“software”);
        string      s2(“SoftWare”);

        if (s1 == s2)
                cout << “equal.” << endl;
        else
                cout << “not equal.” << endl;

        s2 = s1.substr(4, 4);
        cout << s2;

        string      s3 = s1 + s2;
        cout << s3 << endl;
}
```

Thư viện C++

- Lớp vector:
  - File header <vector>.
  - Lớp mảng kiểu T.
  - **Giải quyết 3 vấn đề con trỏ.**
  - Các phương thức chính:
    - vector<T>(): khởi tạo mảng kiểu T.
    - size(): lấy kích thước mảng.
    - push_back(T): thêm phần tử vào cuối mảng.
    - Toán tử [ ]: lấy phần tử tại một vị trí nào đó.
  - Tham khảo trang 60 giáo trình HĐT

Thư viện C++

- Ví dụ:

```
void main()
{
        vector<int>     v1;
        v1.push_back(1);
        v1.push_back(2);

        for (int i = 0; i < v1.size(); i++)
                cout << v1[i] << " ";

        vector<PhanSo *>   v2;
        v2.push_back(new PhanSo(2, 6));
        v2[0]->rutGon();
}
```

Tham khảo

- Slide PPLTHĐT của
  - Thầy Nguyễn Minh Huy
  - Thầy Đinh Bá Tiến