Định nghĩa kiểu dữ liệu mới trong C/C++

```
Struct SoPhuc
{
     double Thuc, Ao;
};
void NhapSoPhuc (SoPhuc& a)
{
     cout << "\n Phần thực: "; cin >> a.Thuc;
     cout << "\n Phần ảo: "; cin >> a.Ao;
}
void XuatSoPhuc (SoPhuc a)
{
     Cout << '(' << a.Thuc;
     Cout << ', ' << a.Ao << ')';
}
void main ()
{
     SoPhuc a;
     cout << "Nhập một số phức: ";
     NhapSoPhuc (a);
     cout << "Số phức vừa nhập: ";
```

```
	XuatSoPhuc (a);
}
```

1. Xây dựng tác vụ trên đối tượng dữ liệu kiểu cấu trúc

Khi thực hiện phép gán kiểu dữ liệu cấu trúc cho 1 kiểu dữ liệu cấu trúc khác, hệ thống sao chép y từng thành phần.

Nếu không có tác vụ cộng thì phép cộng hai số phức được thực hiện như sau.

```
void main ()
{
	SoPhuc a, b, c;
	cout << "Nhập một số phức: ";
	NhapSoPhuc (a);
	cout << "Nhập một số phức: ";
	NhapSoPhuc (b);
	c.thuc = a.Thuc + b.Thuc;
	c.Ao = a.Ao + b.Ao;
	cout << "Tổng số phức: " << XuatSoPhuc(c);
}
```

2. Xây dựng tác vụ cộng

```
SoPhuc CongSoPhuc (SoPhuc a, SoPhuc b)
{
	SoPhuc c;
	c.Thuc = a.Thuc + b.Thuc;
	c.Ao = a.Ao + b.Ao;
```

```
	return c;
}
void main ()
{
	SoPhuc a, b, c;
	cout << "Nhập một số phức: ";
	NhapSoPhuc (a);
	cout << "Nhập một số phức: ";
	NhapSoPhuc (b);
	c = CongSoPhuc (a,b);
	cout << "Tổng số phức: " << XuatSoPhuc(c);
}
```

3. Định nghĩa tác vụ như một thành phần của cấu trúc

```
Struct SoPhuc
{
	double Thuc, Ao;

	SoPhuc Cong (SoPhuc b) const;
	void Nhap ();
	void Xuat () const;
};
void SoPhuc::Nhap ()
{
```

```
      cout << "\n Phần thực: "; cin >> Thuc;
      cout << "\n Phần ảo: "; cin >> Ao;
}
void SoPhuc::Xuat ()const
{
      Cout << '(' << Thuc;
      Cout << ', ' << Ao << ')';
}
SoPhuc  SoPhuc::Cong (SoPhuc b)const
{
      SoPhuc kq;
      Kq.Thuc = Thuc + b.Thuc;
      Kq.Ao = Ao + b.Ao;
      return kq;
}
void main ()
{
      SoPhuc a, b, c;
      cout << "Nhập một số phức: ";
      a.Nhap();
      cout << "Nhập một số phức: ";
      b.Nhap();
      C = a.Cong(b);
```

```
	cout << "Tổng số phức: " << c.Xuat();
}
```

4. Phương thức thiết lập

- Là phương thức thiết lập đối tượng dữ liệu và khởi tạo giá trị cho các dữ liệu thành phần.
- Được tự động gọi khi có sự khai báo đối tượng dữ liệu thuộc kiểu.
- Tên trùng tên kiểu, không có kiểu dữ liệu trả về.
- PTTL có thể có hoặc không có tham số.
    - Đối với 1 kiểu dữ liệu cấu trúc có thể có nhiều PTTL.
    - PTTL không có tham số gọi là PTTL mặc định.
    - Nếu không có định nghĩa PTTL nào cho kiểu dữ liệu cấu trúc thì trình biên dịch sẽ cung cấp PTTL mặc định.

```
Struct SoPhuc
{
	double Thuc, Ao;

	SoPhuc ();
	SoPhuc (double T, double A);
	//SoPhuc (double T = 0.0, double A = 0.0);
};
SoPhuc::SoPhuc ()
{
	Thuc = 0.0;
	Ao = 0.0;
}
SoPhuc::SoPhuc (double T, double A)
```

```
{
    Thuc = T;
    Ao = A;
}
```

5. Con trỏ Pointer

- Con trỏ là đối tượng dữ liệu mà giá trị (nội dung) của nó là địa chỉ của một đối tượng dữ liệu khác.
- Hiệu quả trong trường hợp những vùng nhớ không được tạo ra trước trong thời gian thực thi.

Khai báo

int* p; // biến p là một con trỏ trỏ đến ĐTDL kiểu int

VD:

```
int n;
int* const p = &n; // con trỏ hằng, nội dung con trỏ này không thay đổi được
```

Các phép toán trên con trỏ

- Phép gán: hai con trỏ phải cùng kiểu.
- Cộng: cộng ĐTDL con trỏ với ĐTDL số nguyên

```
int* p; // VD p chỉ đến địa chỉ 400
p + 1; //tạo ra con trỏ mới trỏ đến địa chỉ 401
```

- Trừ
    - Trừ ĐTDL con trỏ với ĐTDL số nguyên
    - 2 ĐTDL con trỏ cho biết khoảng cách giữa hai vùng nhớ là bao nhiêu.
- **Toán tử lấy địa chỉ**

  & <tên ĐTDL>

  Tương thích

  <kiểu của ĐTDL>*

VD:

```
int n;
int* p;
p = &n;
```

- Truy xuất ĐTDL do con trỏ trỏ đến

  * <tên con trỏ>

  VD:

```
int n = 5;
int* p = &n;
Cout << *p << p; // xuất 5    0x400
```

Con trỏ đến ĐTDL kiểu cấu trúc

<tên con trỏ> → <tên thành phần>

*<tên con trỏ>.<tên thành phần>

VD:

```
PhanSo a;
PhanSo* p = &a;
cout << p → TuSo;
p → Xuat();
a.Xuat();
```

6. Vùng nhớ cấp phát động

- Kích thước vùng nhớ được cấp phát vào thời gian thực thi chương trình.
- Hệ thống cấp phát trả về địa chỉ vùng nhớ, có thể dùng con trỏ để giữ lại địa chỉ này.

Xin cấp phát vùng nhớ

new <tên kiểu DL>; // xin cấp phát vùng nhớ mà vùng nhớ đó đủ chứa kiểu dữ liệu

```
int* p = new int;
```

hay new <tên kiểu dữ liệu> [<số phần tử>]; // xin cấp phát vùng nhớ đủ chứa số phần tử, mỗi phần tử có kiểu dữ liệu

```
int* p = new int[10];
```

- Nếu cấp phát thất bại con trỏ trả về con trỏ NULL.
- Khi không sử dụng phải giải phóng vùng nhớ

delete <tên con trỏ>;

delete[] <tên con trỏ>;

VD:

```
int* p = new int;
delete p;
int* p = new int[10];
delete []p;
```

- **Xây dựng kiểu dữ liệu mảng động**

```
typdef int ItemType;
struct DArray
{
	ItemType* p;//con trỏ chỉ đến vùng nhớ chứa các phần tử
	int Size;//kích thước hiện tại của vùng nhớ
	int Num; //số phần tử hiện có

	DArray(int sz = 10);
	DArray(const ItemType a[], int n);

	void Input();
	int InsertLast (const ItemType& x);
	int IsFull()const {return Num >= Size;}
	int IsEmpty()const {return Num == 0;}
	int GrowBy (int sz = 10);

	~DArray() {delete[] p;}
```

```
        DArray(const DArray& b);
        Array& operator=(const DArray& b);
};
DArray::DArray(int sz = 10)
{
        p = new ItemType [Size = sz];
        if (p == NULL)
              return;
        Num = 0;
}
DArray::DArray (const ItemType a[], int n)
{
        p = new ItemType [Size = n];
        memcpy (p, a, Size * sizeof (ItemType));
        Num = n;
}
```

- **Input**

- Mảng đang xét đang có quản lý một vùng nhớ để lưu giữ các phần tử.
- Nếu số lượng phần tử cần nhập vượt quá kích thước vùng nhớ đang có thì:
    - Giải phóng vùng nhớ cũ.
    - Xin cấp phát vùng nhớ mới đủ chứa số phần tử sử dụng.

- **InsertLast (Mảng đầy)**

- Xin cấp phát vùng nhớ mới lớn hơn.
- Sao chép nội dung từ vùng nhớ cũ sang vùng nhớ tạm.
- Xóa vùng nhớ cũ.
- Trỏ đến vùng nhớ mới.
- Thêm phần tử vào cuối vùng nhớ mới.

```
void DArray::Input()
{
   If (IsFull())
```

```
	{
		p = new ItemType [Size = Num];
	}
	If p == NULL)
	{
	   Cout << "Không cấp phát được vùng nhớ."
	   return;
	}
	for (int i = 0; I < Num; i++)
	{
		cout << "Phần tử [" << i << "]";
		cin >> p[i];
	}
}
int Darray::GrowBy (int sz)
{
	ItemType *tmp = new ItemType[Size + sz];
	if (!p) return 0;
	memcpy (tmp, p, Size * sizeof (ItemType);
	delete[] p;
	p = tmp;
	Size += sz;
	return 1;
```

```
}

int Darray::InsertLast (const ItemType & x)

{

     If (IsFull() && !GrowBy()) return 0;

     p[Num++] = x;

     return 1;

}
```

Phương thức thiết lập bản sao

Sử dụng trong 02 trường hợp

- Truyền thông số bằng trị
- Kết quả trả về

```
DArray::DArray (const DArray& b)

{

     p = new ItemType [Size = b.Size];

     If (!p)

     {

          cout << "Không thể cấp phát vùng nhớ.";

          return;

     }

     Num = b.Num;

     memcpy (p, b.p, Num * sizeof (ItemType);

}
```

Phép gán

- Khi thực hiện phép gán kiểu dữ liệu cấu trúc cho 1 kiểu dữ liệu cấu trúc khác, hệ thống sao chép y từng thành phần.
- Phép gán trong con trỏ (mảng động) dẫn đến cái sai.
    - Không giải phóng vùng nhớ, vùng nhớ cũ lơ lửng.
    - Trỏ đến vùng nhớ không xác định.

```
a = b;
```

- Kích thước vùng nhớ cấp phát động được quản lý bởi a không đủ chứa nội dung vùng nhớ cấp phát động được quản lý bởi b.
    - Giải phóng vùng nhớ do a quản lý.
    - Xin cấp phát vùng nhớ mới đủ lớn cho a.
    - Sao chép nội dung từ vùng nhớ do b quản lý sang vùng nhớ do a quản lý.
    - Các thành phần tĩnh gán bình thường.

```
DArray& DArray::operator=(const DArray& b)
{
	if (this != &b)
		delete []p;
	p = new ItemType [Size = b.Size];
	if (!p)
	{
		cout << "Không thể cấp phát vùng nhớ.";
		return;
	}
	Num = b.Num;
	memcpy (p, b.p, Num * sizeof(ItemType);
	return *this;
```

```
}
```