Phương pháp Lập trình Hướng đối tượng

Lớp Đối tượng

"Class = Attrib + Method + Priv"

GV: Lê Xuân Định

Nhắc lại chuyện xưa – Struct

- Bạn đã gặp những struct nào?
- Nếu không dùng struct có được không?
  - Về lý thuyết, mọi bài toán đều giải được không cần struct!
- Tại sao phải đóng gói thành 1 struct?
  - Rõ ràng: Làm chương trình ngắn gọn, dễ đọc, gần với thực tế;
  - Tiện dụng: Những dữ liệu đi chung được quản lý chung;
  - Tăng tính module: Dễ tái sử dụng struct và các hàm xử lý struct cho bài toán quản lý khác; Dễ thay đổi các trường dữ liệu mà không làm ảnh hưởng đến các hàm quản lý chung.

"Điểm tổng kết 3 SV"
Tính module của Struct

Hãy viết chương trình cho phép **nhập điểm** (lý thuyết, thực hành) của ba SV từ bàn phím, và **xuất ra màn hình điểm tổng kết**.

```
struct SinhVien{
    float dLT;
    float dTH;
};
typedef struct SinhVien SINHVIEN;

void Nhap(SINHVIEN & sv);
void XuatDTK(SINHVIEN  sv);
```

Do hàm main() sử dụng **struct SinhVien** như một ***đơn vị (không đụng vào từng thành phần của struct)*** nên...

```
void main(){
    SINHVIEN An, Binh, Chi;
    Nhap(An); Nhap(Binh); Nhap(Chi);
    XuatDTK(An); XuatDTK(Binh); XuatDTK(Chi);
}
```

"Điểm tổng kết 3 SV"
Tính module của Struct

Hãy viết chương trình cho phép **nhập điểm** (lý thuyết, thực hành, điểm cộng) của ba SV từ bàn phím, và **xuất ra màn hình điểm tổng kết**.

```
struct SinhVien{
    float dLT, dTH;
    float dCong;
};
typedef struct SinhVien SINHVIEN;

void Nhap(SINHVIEN & sv);
void XuatDTK(SINHVIEN  sv);
```

Do hàm main() sử dụng **struct SinhVien** như một ***đơn vị (không đụng vào từng thành phần của struct)*** nên...

```
void main(){
    SINHVIEN An, Binh, Chi;
    Nhap(An); Nhap(Binh); Nhap(Chi);
    XuatDTK(An); XuatDTK(Binh); XuatDTK(Chi);
}
```

"Đóng gói Mảng"
Tính tiện dụng của Struct

- Đặt vấn đề: Kiểu "**mảng**" trong C/Pascal là một ***kiểu dữ liệu hoàn chỉnh*** hay không?
  - Các thao tác trên mảng chỉ cần mảng?
  - Muốn sao chép mảng, phải sao chép từng phần tử.
- Giải quyết: Đóng gói **mảng *a*[]** và **số phần tử *n*** thành một struct

```
struct ArrayT{ T a[…];  int n; };
```

- Lợi ích: Đối xử với toàn mảng như 1 đơn vị dữ liệu (1 biến)
  - Truyền tham số: Chỉ cần 1 tham số → Tránh trường hợp quên truyền số phần tử (*n*).
  - Sao chép mảng: Chỉ một phép gán (Không cần for()).

2 Đặc trưng của việc Đóng gói

"Những thứ thường/luôn đi chung với nhau thì gom lại thành một gói."

- ➔ Tiện dụng: Đối xử với chúng như ***1 đơn vị***.

"Người sử dụng gói không cần quan tâm đến cấu trúc bên trong của gói."

- ➔ Tính module: Quản lý gói có thể ***độc lập*** với xử lý dữ liệu trong gói.

Đóng gói trong Hướng đối tượng

"Những thứ thường/luôn đi chung với nhau thì gom lại thành một gói."

- ➔ Tiện dụng: Đối xử với chúng như ***1 đơn vị***.

"Người sử dụng gói không được quan tâm đến cấu trúc & xử lý bên trong của gói."

- ➔ Tính module: Bên sử dụng gói ***độc lập*** với bên xử lý dữ liệu trong gói.

Áp dụng 2 nguyên tắc này cho cả các hàm (chứ không chỉ cho dữ liệu như struct), ta có "*Phương pháp Lập trình* ***Hướng đối tượng***"!

Thảo luận về "Đóng gói"

"Những thứ thường/luôn đi chung với nhau thì gom lại thành một gói."

- ➔ Tiện dụng: Đối xử với chúng như ***1 đơn vị***.

"Người sử dụng gói không được quan tâm đến cấu trúc & xử lý bên trong của gói."

- ➔ Tính module: Bên sử dụng gói ***độc lập*** với bên xử lý dữ liệu trong gói.

Trong (biến, thư viện, câu lệnh, mảng, struct, hàm), những cái nào là "gói" của những cái nào? Các "gói" đó tiện dụng ra sao? Những "gói" nào có tính module?

Họ các Hàm Xử lý Chuỗi

```
char s[256];
```

Những thứ luôn đi chung với nhau

```
int   strlen(                    );
void  strcat(           char str[]);
int   strcmp(           char str[]);
char* strstr(           char str[]);
```


- Các **hàm xử lý chuỗi** trong thư viện “string.h”
  - Đều gắn liền với 1 tham số chuỗi (***chuỗi bị xử lý***)

Họ các Hàm Xử lý Chuỗi

string
`char s[256];`

`int strlen( );`
`void strcat( char str[]);`
`int strcmp( char str[]);`
`char* strstr( char str[]);`

Những thứ luôn đi chung với nhau thì gom lại thành một gói

- Các **hàm xử lý chuỗi** trong thư viện "string.h"
  - Đều gắn liền với 1 tham số chuỗi (***chuỗi bị xử lý***)
- Ta gom các hàm đó và ***chuỗi bị xử lý*** lại thành 1 gói
  - Tạo thành một **kiểu mới** (gọi là ***lớp***): `string`

“Lớp” – Kiểu đối tượng

string

```
char buffer[256];   int size;   ...

int  length (                    );
void append (          string str);
int  compare(          string str);
int  find   (          string str);
```


Để đảm bảo **tính module,** phần thuộc tính là **hộp đen**: Người sử dụng không cần và không được quan tâm đến cấu trúc dữ liệu!

- Các đối tượng thuộc lớp `string` (biến có kiểu `string`)
  - Không chỉ chứa chuỗi ký tự (`char[]`), mà còn chứa...
  - Những thuộc tính (dữ liệu) liên quan như `size`, v.v.
  - Cùng các phương thức (hàm) liên quan.

“Lớp” – Dữ liệu & Hành động

Đóng gói tất cả các **hàm** xử lý cùng một loại dữ liệu và các **biến** dữ liệu đó vào thành một **lớp** (*kiểu mới*)

- **Thành phần Dữ liệu**: Các thuộc tính (biến nội bộ)
  -
- **Thành phần Hành động**: Các phương thức (hàm) *xử lý dữ liệu (các thuộc tính)*.
  -

```
class string
{
private:
  char buffer[256];
  ...
public:
  string();
  string(char initStr[]);
  int length();
  bool empty();
  char at(int pos);
  void append(string str);
  void insert(int pos,
              string str);
  int compare(string str);
  int find(string str);
  ...
};
```

Khai báo lớp

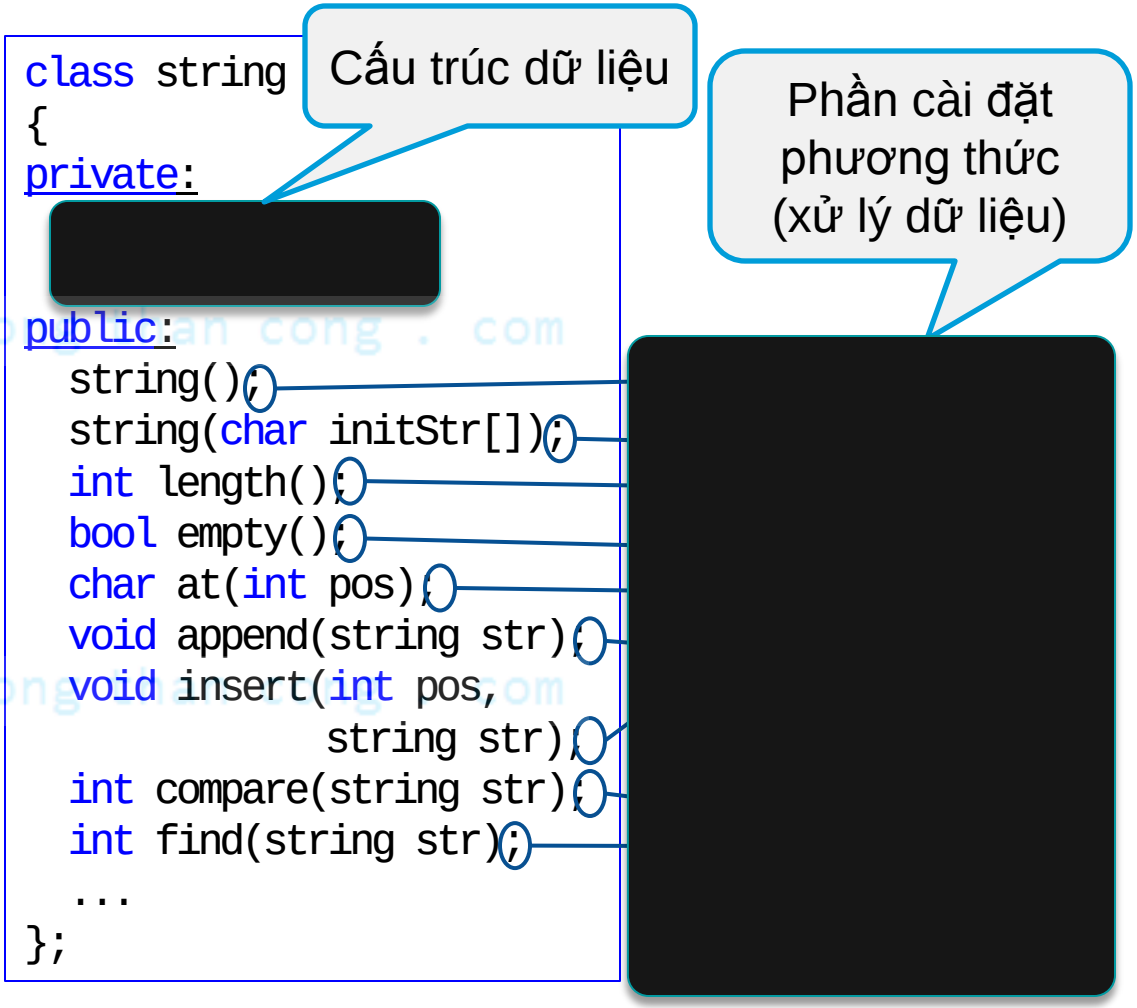
string
buffer: char[] ...
string() string(initString) length(): int empty(): bool at(pos): char append(str) insert(pos, str) erase(pos, n) compare(str): int find(str): int substr(pos,n): string ...

Sơ đồ UML

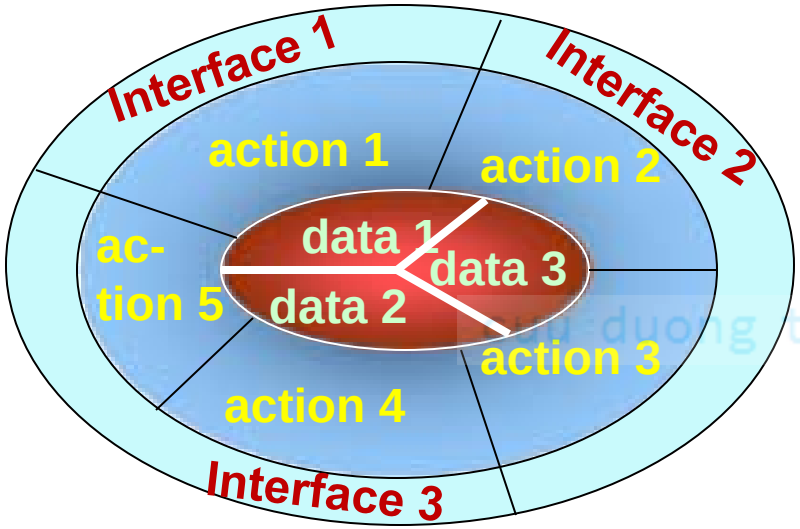
“Lớp” – Dữ liệu & Hành động

Đóng gói tất cả các **hàm** xử lý cùng một loại dữ liệu và các **biến** dữ liệu đó vào thành một **lớp** (*kiểu mới*)

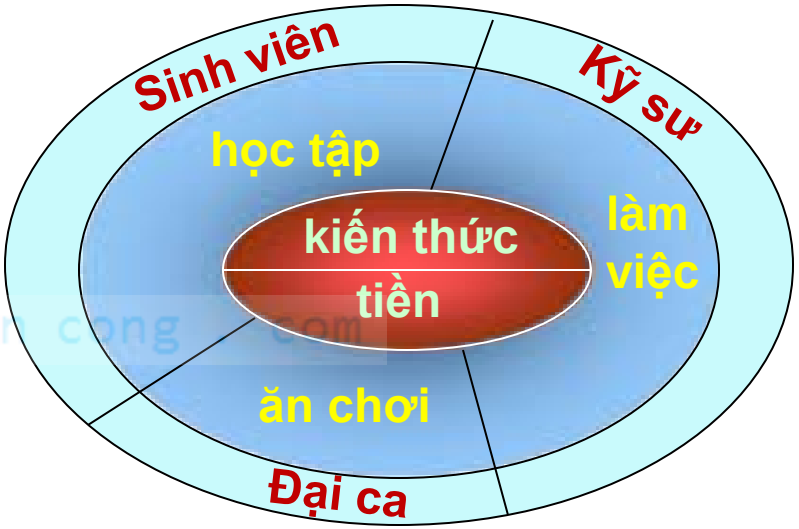
- **Thành phần Dữ liệu**: Các thuộc tính (biến nội bộ)
  - Thường là “private”, chỉ dùng trong nội bộ đ.tượng
- **Thành phần Hành động**: Các phương thức (hàm) *xử lý dữ liệu (các thuộc tính)*.
  - Thường là “public” cho người khác sử dụng.
  - Nhưng *phần cài đặt vẫn của riêng đ.tượng!*



“Lớp” – Đóng gói kín



Đối tượng tổng quát

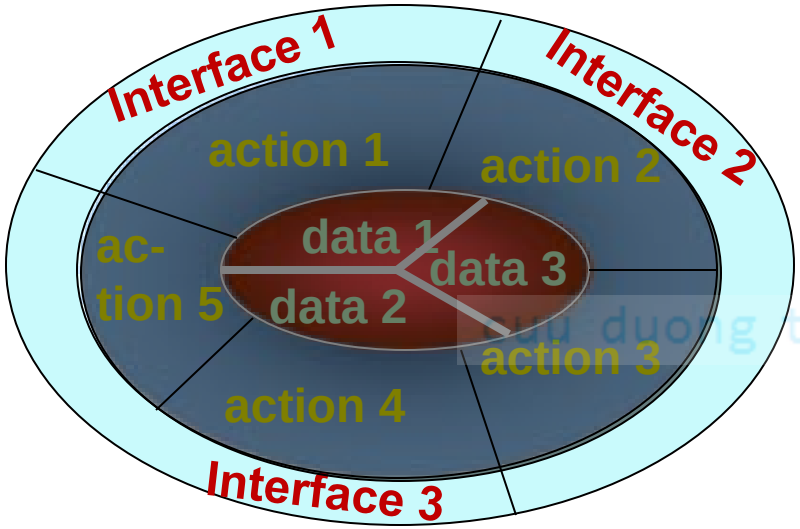


Đối tượng Kỹ sư

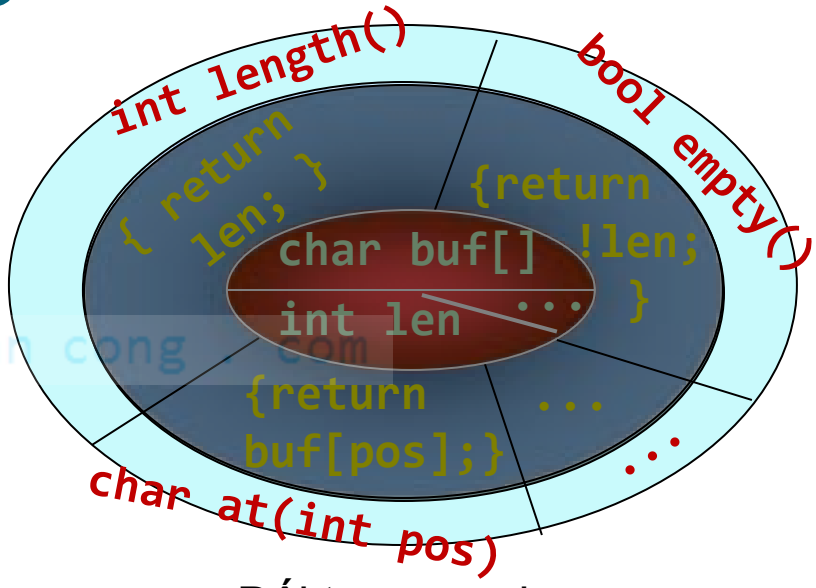
- Mỗi đối tượng được cấu thành từ 3 phần:
  - Nội dung dữ liệu: Các thuộc tính bên trong, được bao bọc bởi
  - Thành phần hành động: Các phương thức xử lý dữ liệu;

    Mỗi p.thức gồm
    - Phần cài đặt p.thức: Định nghĩa cách hoạt động của phương thức, và
    - Phần giao diện: Lớp vỏ ngoài cùng để giao tiếp với thế giới bên ngoài.

“Lớp” – Đóng gói kín



Đối tượng tổng quát



Đối tượng string

- Trên quan điểm sử dụng (nhìn từ ngoài vào):
  - *Chỉ thấy **giao diện*** của đối tượng
  - Không thấy phần cài đặt bên trong:
    - Không thấy phần cài đặt phương thức (cách xử lý dữ liệu);
    - Không thấy cấu trúc dữ liệu (các thuộc tính).

“Lớp” – Mở rộng kiểu Cấu trúc

- Định nghĩa ra **kiểu mới**
  - Mỗi lớp/cấu-trúc là 1 kiểu tự tạo, hoàn toàn mới.
  - Dùng kiểu tự tạo đó để **khai báo (tạo ra) biến**

```
struct SinhVien sv, a, b, c;
string str, q, r, t;
```

  - “Lớp” hơn “cấu trúc” ở thành phần **hành động** (hàm).

- **Truy cập các thành phần** của một “gói” (đối tượng, cấu trúc) qua **toán tử “của”** (“**.**” hoặc “**->**” với con trỏ)
  - Thành phần dữ liệu:

```
float d = sv.dLT //Đọc điểm lý thuyết của sv
```

  - Thành phần hành động:

```
int l = str.length(); //Gọi p.thức tính độ dài của str.
```

Ví dụ Sử dụng Đối tượng string

Chương trình nhập một chuỗi, tính độ dài của nó và xuất kết quả.

- "`string`" là một kiểu mới → khai báo biến: `string s;`
- Gọi phương thức "tính độ dài": `s.length();`
- *Hoàn toàn không đụng tới nội dung dữ liệu chứa trong s.*

```
#include <iostream>
#include <string>
using namespace std;

void main()
{
  string s;
  cout<<"do dai = "<<s.length()<<endl;
  cout<<"nhap chuoi moi: ";
  cin>>s;
  cout<<"do dai cua \""<<s<<"\" = "
      <<s.length()<<endl;
  fflush(stdin); cin.get();
}
```

Hoạt cảnh Sử dụng Đối tượng

- Viết CTrình quản lý tập số nguyên như sau:
  - Tạo tập rỗng; Thêm vào tập các số được nhập từ đ.tượng NhapSo
  - Xuất ra MH số lượng phần tử đã thêm thành công vào tập số;
  - Liệt kê các phần tử trong tập theo thứ tự tự nhiên hoặc tăng dần.

Ø Sử dụng 2 lớp đối tượng được cài đặt sẵn với giao diện như sau:

```
class TapSo
  { public:
     TapSo();
     int SoLuong();
     void Them(int x);
     void LietKe(bool tang);
     ...
  };
```

```
class NhapSo
{ public:
   NhapSo();
   bool ConNua();
   int Tiep();
   ...
}; //Nhập các số: 2, 7, 15,
  // 3, 5, 9, 5, 4, 3, -1, 6, 1, 8
```

Hoạt cảnh Sử dụng Đối tượng

```
class NhapSo
{ public:
  NhapSo(); //Chuẩn bị nhập các số nguyên
  bool ConNua(); //Còn số để nhập tiếp không?
  int Tiep(); //Trả về số được nhập tiếp theo
  ... };
```

```
class TapSo
{ public:
  TapSo(); //Tập rỗng
  int SoLuong(); //Số phần tử trong tập
  void Them(int x); //Thêm x >0 và chưa có trong tập
  void LietKe(bool tang); //In ra màn hình các phần tử
        // theo thứ tự tăng dần nếu tang==true, hoặc theo tt tự nhiên
  ... };
```

BT Ứng dụng

- Hãy khai báo cho các lớp được sử dụng trong đoạn code sau: (Phần nào không thể biết thì để ba chấm)

```
bool testClasses(){
    XYZ o(123.4, "test");
    Temp t, *p = new Temp(&o);
    t.setAt(1, 12.3);
    if(!t.equals(p)){
          t.setAt(1, p->getAt(2));
    }
    o.value(5.0);
    return (o.value()==5.0);
    delete p;
}
```

Thực thể hoá

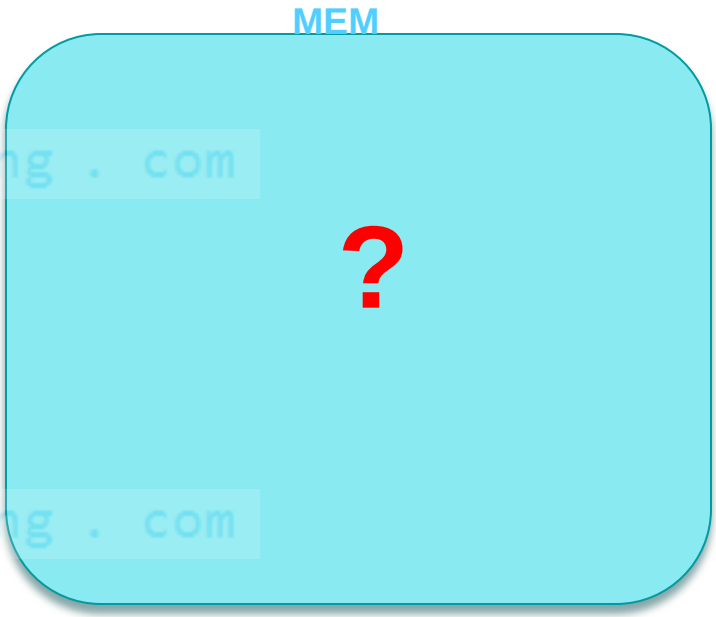
"Kiểu là khuôn mẫu để in ra các thực thể"

cuu duong than cong . com

Các kiểu dữ liệu Phức hợp

```
struct SMix{int i; float t;};
```

Khi kết thúc câu khai báo này, trong bộ nhớ dữ liệu có gì???

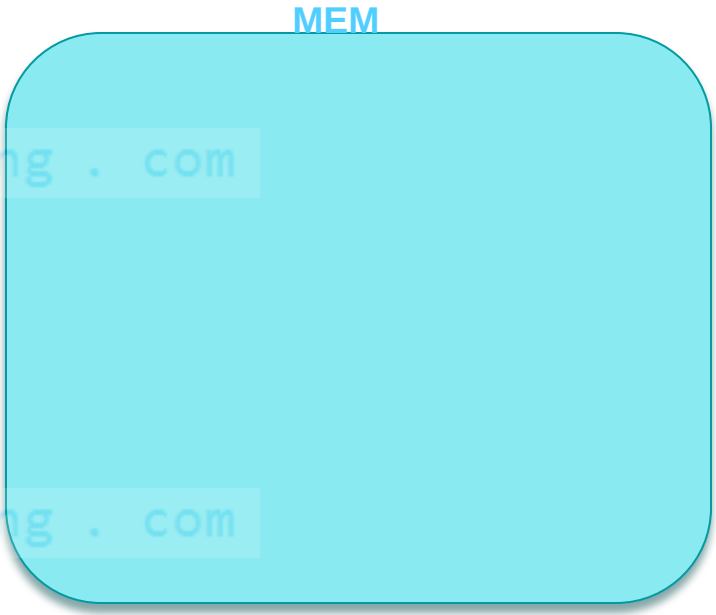


Các kiểu dữ liệu Phức hợp

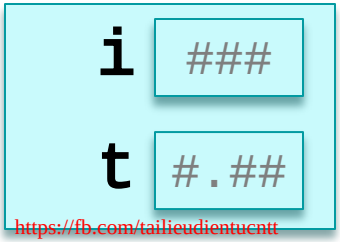
- `struct SMix{int i; float t;};` **`int[3]`**

Vậy khi nào thì bộ nhớ mới bị tác động (có dữ liệu)?

- Kiểu dữ liệu là khái niệm logic nằm ***ngoài bộ nhớ*** *dữ liệu*.
  - Khi định nghĩa kiểu, bộ nhớ dữ liệu vẫn còn trống.



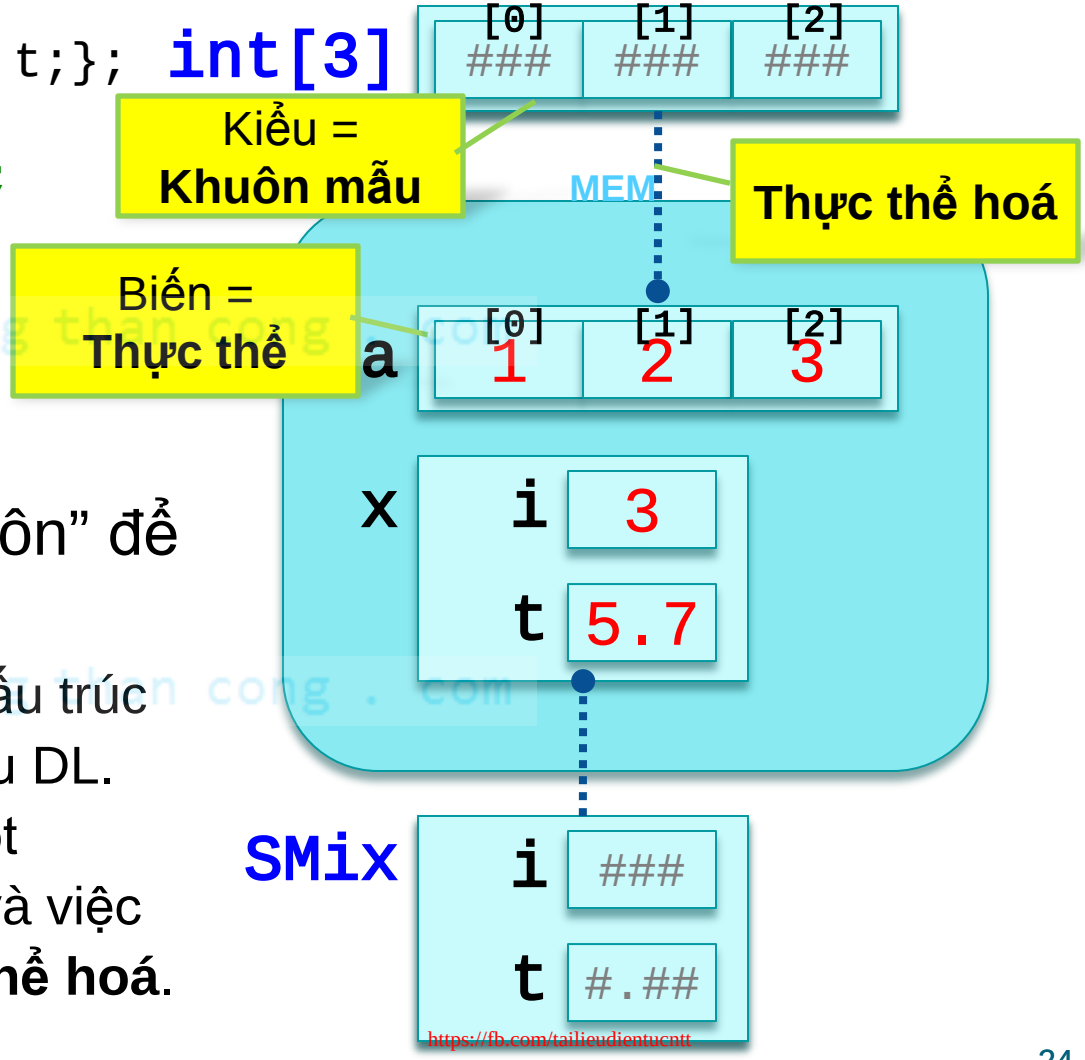
SMix



Các kiểu dữ liệu Phức hợp

```
struct SMix{int i; float t;};
void main(){
  //int[3] a = {1, 2, 3};
  int a[3] = {1, 2, 3};
  SMix x = {3, 5.7};
  a[1] = x.i;
  x.t = a[2];
}
```

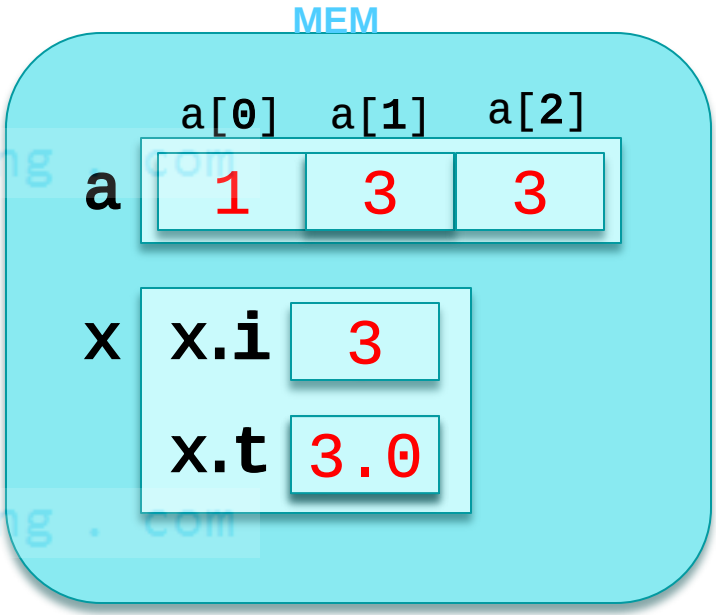
- Kiểu dữ liệu là cái "khuôn" để "đúc" ra các biến.
  - Vùng nhớ của biến có cấu trúc được định nghĩa bởi kiểu DL.
  - Mỗi *biến cụ thể* đó là một ***thực thể*** *của kiểu DL*, và việc "đúc" ra chúng là **thực thể hoá**.



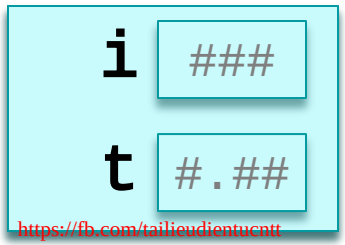
Các kiểu dữ liệu Phức hợp

```
struct SMix{int i; float t;};
void main(){
  //int[3] a = {1, 2, 3};
  int a[3] = {1, 2, 3};
  SMix x = {3, 5.7};
  a[1] = x.i;
  x.t = a[2];
}
```

int[3]



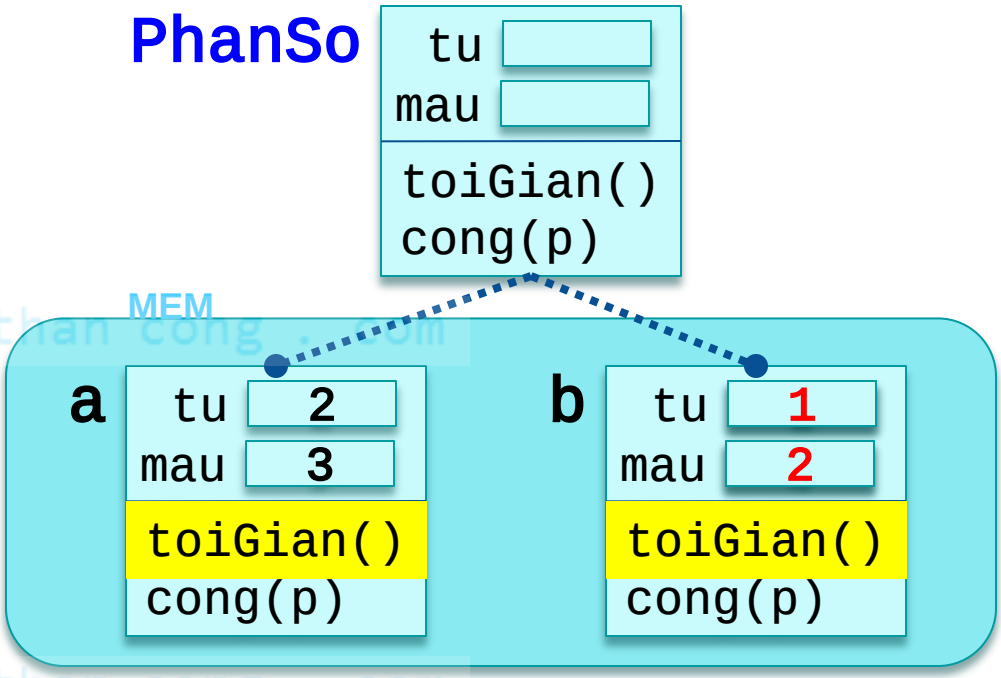
SMix



- **Mỗi phần tử** của mảng và **mỗi thành phần** của cấu trúc là **một *biến***.
  - Sử dụng hoàn toàn giống biến bình thường.
  - Có tên `a[0]`, `a[1]`, `a[3]` và `x.i`, `x.t` (gắn cố định với biến).

Mỗi Lớp là 1 Kiểu mới

```
class PhanSo{ int tu, mau;
public:
  void toiGian();
  PhanSo cong(PhanSo p);
  ...
};
void main(){
  PhanSo a(2,3);
  PhanSo b(2,4);
  a.toiGian();
  b.toiGian();
}
```



- **Mỗi thực thể** đối tượng có một bộ các *hành động* cùng với *dữ liệu (trạng thái)* của ***riêng*** mình.
  - Cùng một phương thức, nhưng *mỗi thực thể khác nhau sẽ* ***hành động khác nhau*** → *Tác động khác nhau* → *Kết quả khác nhau*.

Kiểu – Khuôn mẫu của Biến

- Định nghĩa kiểu
  - Định nghĩa "khuôn mẫu" cho các biến tương ứng
    - *Kích thước & cấu trúc* vùng nhớ
    - Loại & *miền xác định* của giá trị
  - Những cái "khuôn" này nằm ***ngoài bộ nhớ*** *dữ liệu*.
  - Các kiểu cơ bản được *định nghĩa sẵn*.
  - Các kiểu mảng được *định nghĩa tự động* ngay khi được dùng tới.
  - Các kiểu `struct` & `class` được LTV *tự định nghĩa* trong c.trình.
- Thực thể hoá
  - Mỗi biến được tạo ra bằng cách "dập khuôn" từ kiểu tương ứng (thông qua: Khai báo biến, cấp phát động).
  - Mỗi biến đó là một ***thực thể*** *của kiểu* tương ứng.
    - Xác định *địa chỉ* vùng nhớ, *giá trị* (và *tên biến* tĩnh).

Ví dụ Ứng dụng

```
int tu=0, mau=1, k=0, t=0;
class PhanSo {
private: int tu, mau;
public: PhanSo(); // 0/1
  void nhan(int k);
};
```

```
PhanSo::PhanSo():tu(0),mau(1){}
void PhanSo::nhan(int k){
  int t = tu * k;
  tu = t;
}
```

```
void main(){
  int tu=0, mau=1, k=0, t=0;
  /* 1) Khi thực hiện tới dòng này, trong bộ nhớ dữ liệu
  có những biến nào? Hãy vẽ những biến đó ra cùng với giá
  trị đang chứa (nếu là "rác" thì để trống).
     2) Nếu còn biến nào đã được khai báo (bên trên) mà
  chưa có trong bộ nhớ thì hãy viết thêm lệnh vào dưới
  dòng chú thích này để chúng xuất hiện trong bộ nhớ.*/
}
```

Ví dụ Ứng dụng

```
int tu=0, mau=1, k=0, t=0;
class PhanSo {
private: int tu, mau;
public: PhanSo(); // 0/1
  void nhan(int k);
};
```

```
PhanSo::PhanSo():tu(0),mau(1){}
void PhanSo::nhan(int k){
  int t = this->tu * k;
  this->tu = t;
}
```

```
void main(){
  int tu=0, mau=1, k=0, t=0;
  /* 1) Khi thực hiện tới dòng này, trong bộ nhớ dữ liệu
  có 4 biến toàn cục ::tu, ::mau, ::k, ::t, và 4 biến cục
  bộ main().tu, main().mau, main().k, main().t*/
  PhanSo p, q; // 2) Tới đây mới thêm p.tu, p.mau, q.tu, q.mau;
  p.nhan(3)/* 2') Tới đây (trước khi kết thúc hàm) mới có
   p.nhan(3).k, p.nhan(3).t.*/ ;
}
```

Cài đặt Lớp

"An toàn là trên hết!"

Bài toán Mẫu: "Điểm tổng kết 3 SV"

- Hãy viết chương trình cho 3 SV **đi thi** (lấy điểm lý thuyết, thực hành) và **tính điểm tổng kết** của từng SV.
  - **đTK** = (đLT*6 + đTH*4) / 10
  - **Điểm LT** và **điểm TH** của SV chỉ có được thông qua hành động "***thi LT***", "***thi TH***". (Muốn thay đổi thì phải "thi lại", tức thực hiện hành động "thi" một lần nữa.)
    - Hành động "thi": Yêu cầu GV nhập điểm tương ứng từ bàn phím.

Khai báo lớp

```
class SinhVien
  {
  private:
       float dLyThuyet;
       float dThucHanh;
  public:
       SinhVien();
       //~SinhVien();
       void thiLT();
       void thiTH();
       float tinhDTK();
  private:
       float nhapDiem(string loaiDiem);
  };
```

Mọi thuộc tính đều thuộc vùng ***private***

- Là cấu trúc dữ liệu riêng của đối tượng
- ***Quy tắc đóng gói kín!***

Khai báo lớp

```
class SinhVien
  {
  private:
       float dLyThuyet;
       float dThucHanh;
  public:
       SinhVien();
       //~SinhVien();
       void thiLT();
       void thiTH();
       float tinhDTK();
  private:
       float nhapDiem(string loaiDiem);
  };
```

Hầu hết phương thức thuộc vùng public
- Để cho bên ngoài sử dụng

Một số phương thức thuộc vùng private
- Để dùng riêng trong nội bộ lớp
- Hỗ trợ xử lý cho các phương thức public

Khai báo lớp

```
class SinhVien
  {
  private:
       float dLyThuyet;
       float dThucHanh;
  public:
       SinhVien();
       //~SinhVien();
       void thiLT();
       void thiTH();
       float tinhDTK();
  private:
       float nhapDiem(string loaiDiem);
  };
```

Hàm tạo/huỷ không có kiểu trả về (không ghi "void") và có tên trùng tên lớp

- Hàm tạo được gọi khi tạo đối tượng mới (thực thể hoá)
- Hàm huỷ được gọi khi huỷ đối tượng (kết thúc vòng đời)
  - Tên hàm huỷ có dấu ngã (~) phía trước
  - Lớp bình thường không cần định nghĩa lại hàm huỷ

Khai báo lớp

```
struct SinhVien
   {
   private:
        float dLyThuyet;
        float dThucHanh;
   public:
        SinhVien();
        //~SinhVien();
        void thiLT();
        void thiTH();
        float tinhDTK();
   private:
        float nhapDiem(string loaiDiem);
   };
```

Trong C++, `struct` hoàn toàn tương đương với `class`, chỉ khác nhau ở vùng truy cập mặc định:

- `struct` mặc định là `public`
- `class` mặc định là `private`

Khai báo lớp

```
struct SinhVien
  {
  private:
       float dLyThuyet;
       float dThucHanh;
  public:
       SinhVien();
       //~SinhVien();
       void thiLT();
       void thiTH();
       float tinhDTK();
  private:
       float nhapDiem
  (string loaiDiem);
  };
```

```
struct SinhVien
  {
       SinhVien();
       //~SinhVien();
       void thiLT();
       void thiTH();
       float tinhDTK();

  private:
       float nhapDiem
  (string loaiDiem);
       float dLyThuyet;
       float dThucHanh;
  };
```

Cài đặt phương thức

- Nguyên mẫu hàm bên ngoài khối khai báo lớp
  - Thêm tên_lớp: phía trước tên phương thức. ("**::**" = "**của**")
- Thân phương thức, giống như thân hàm thường
  - Truy cập Thuộc tính & phương thức của ***đối tượng đang thực hiện hành động*** thông qua ***con trỏ this***
    this->thuocTinh**, this->**phuongThuc()
- VD:

```
float SinhVien::tinhDTK() //PThức "tính ĐTK" của lớp SinhVien
{
  float dtk = (this->dLyThuyet*6 + this->dThucHanh*4)/10;
  return dtk;
}
```

Nói về tôi...

- Tôi, I, watashi, je, wo, ngo, … self, this
  - Là **con trỏ** đến ***đối tượng đang thực hiện hành động*** (phương thức đang được cài đặt).
- Ý thức "của tôi"
  - Thuộc tính của tôi, VD: void setX(int x){ this->x = x; }

    ➔ Phân biệt với tham số, biến cục bộ
  - Hành động của tôi, VD:

```
void Fraction::increase(Fraction d){ // this += d
   Fraction t = this->add(d); // t = this + d
   this->set(t);  // this = t
}
```

    ➔ Phân biệt với hàm toàn cục
- Khi cần "nói về tôi"

Nói về tôi...

- Tôi, I, watashi, je, wo, ngo, … self, this
  - Là **con trỏ** đến ***đối tượng đang thực hiện hành động*** (phương thức đang được cài đặt).
- Ý thức "của tôi"
- Khi cần "nói về tôi"
  - Truyền tham số, VD:

```
SinhVien::SinhVien(DaiHoc u){ // Tạo SV mới trong đại học u
    ...
    u.add(this); // Thêm SV này vào danh sách SV của u
}
```

  - Lấy địa chỉ, VD:

```
SinhVien::SinhVien(){ this->mssv = (int)this; ... }
```

  - ➔ Phân biệt với thực thể khác (trong cùng lớp)

Khởi tạo giá trị Thuộc tính

- Mọi thuộc tính đều phải được khởi tạo giá trị
  - Hoặc bằng ***giá trị mặc định*** trong khai báo lớp;
  - Hoặc thông qua các hàm tạo:
    Thường thông qua ***danh sách khởi tạo*** (init list)

```
struct C {
   int i = 100; //default value init.
   int j, k; //not initialized!!!
   C();
   C(int ij);
};
C::C() :j(1), k(10) {
          cout << "default constructor" << endl;
}
C::C(int ij) :j(ij) { //BUG: k has UNSPECIFIED value!!!
          cout << "k = ??? " << this->k << endl;
}
```

Danh sách khởi tạo

Cài đặt hàm Tạo (Constructor)

- ***Mỗi hàm tạo*** đều phải *khởi tạo giá trị cho* ***tất cả các thuộc tính*** *không có giá trị mặc định!*

Vì:

- Một lớp thường có nhiều hàm tạo khác nhau, nhưng *mỗi đối tượng chỉ được tạo ra đúng 1 lần bởi một trong các hàm tạo của lớp*.
  - "Sinh ra chỉ có một lần, không hai!"
  - Bên sử dụng lớp sẽ chọn ***một*** *trong các hàm tạo* để tạo ra đối tượng:
    - Hàm tạo mặc định (không có tham số), *hoặc*
    - 1 trong các hàm tạo có tham số, *hoặc*
    - Hàm tạo sao chép (có tham số là 1 đối tượng khác cùng lớp).

3 loại hàm Tạo (Constructor)

1. Hàm tạo mặc định: Không có tham số
   - VD sử dụng: `string s, as[10];` //gọi 11 lần hàm `string()`
   - Khởi tạo giá trị mặc định cho tất cả những thuộc tính chưa có giá trị mặc định.
   - Lưu ý: Nếu không định nghĩa hàm tạo nào hết thì C++ sẽ tự định nghĩa 1 *hàm tạo mặc định* ***không làm gì hết***!
     ➔ **Nguy hiểm**!!! Vì các thuộc tính chưa có giá trị mặc định sẽ có giá trị không xác định!
2. Hàm tạo có tham số: Khởi tạo cho tất cả các thuộc tính
   - VD sử dụng: `string s("init value");`
3. Hàm tạo sao chép: Chép từ một đtượng khác cùng lớp
   - VD sử dụng: `string s("init value"), t(s), u = s; f(s);`
   - Đặc biệt sử dụng khi *truyền tham trị* & *trả về giá trị* trong hàm.
   - C++ đã tự định nghĩa hàm tạo sao chép
     ➔ *Nếu không cấp phát động thì* ***không cần định nghĩa lại***.

Tái sử dụng hàm Tạo

- ***Mỗi hàm tạo*** đều phải *khởi tạo giá trị cho* ***tất cả các thuộc tính*** *không có giá trị mặc định!*

➔Nếu có nhiều hàm tạo với số lượng tham số khác nhau thì dễ bị *lặp code*!

➔Giải pháp: Hàm tạo ít tham số có thể tái sử dụng hàm tạo nhiều tham số hơn, bằng cách đưa vào danh sách khởi tạo. VD lớp Hình Chữ Nhật:

```
class Rectangle {
  float width, height;
  bool square = false;
public:
  Rectangle(float w, float h);
  Rectangle(float w);
  ...
};
```

```
Rectangle::Rectangle
(float w, float h)
  : width(w), height(h) {}

Rectangle::Rectangle(float w)
  : Rectangle(w, w) {
  this->square = true;
}
```

Thành phần Tĩnh (static)

- Các thành phần tĩnh của đối tượng
  - Được chỉ định qua từ khoá `static`
  - Còn gọi "*thành phần ở **mức lớp***", tức của chung cả lớp chứ *không của riêng 1 đối tượng* cụ thể nào. (Đối lập với "thành phần ở mức thực thể".)
- Thuộc tính tĩnh: ***Dữ liệu chia sẻ*** chung cho cả lớp
  - Các *hằng* dùng chung cho cả lớp;
  - Các tham số cấu hình lớp, các biến đếm, v.v.
- Phương thức tĩnh: Hàm có thể được ***gọi trực tiếp từ tên lớp "C::f(3)"*** không cần đến đtượng cụ thể.
  - Các thao tác xử lý chung của lớp.

Thuộc tính Tĩnh

- Thuộc tính tĩnh: ***Dữ liệu chia sẻ*** chung cho cả lớp
  - Các *hằng* dùng chung cho cả lớp:
    - Các hằng nguyên được khởi tạo trong khai báo lớp;
      Các hằng không nguyên phải khởi tạo *ngoài* khai báo lớp

```
class C {
        const static int MAX = 9999;
        const static float PI;
}; const float C::PI = 3.1415;
```

  - Các tham số cấu hình lớp, các biến đếm, v.v.

```
class D {
  constexpr static bool debug = true;
  constexpr static int count = 0;
  constexpr static float epsilon = 0.0001;
};
```

Thuộc tính Tĩnh

- Thuộc tính tĩnh: ***Dữ liệu chia sẻ*** chung cho cả lớp
  - Các *hằng* dùng chung cho cả lớp:
  - Các tham số cấu hình lớp, các biến đếm, v.v.

    Với C++ trước C++11 (tương ứng VStudio trước VS2015) thì phải *khởi tạo biến tĩnh* ***trong phần cài đặt lớp ngoài khối định nghĩa lớp*** (trong file .cpp)

```
class D {
  static bool debug;
  static int count;
  static float epsilon;
};
bool D::debug = true;
int D::count = 0;
float D::epsilon = 0.0001;
```

Phương thức Tĩnh

- Phương thức tĩnh: Hàm có thể được ***gọi trực tiếp từ tên lớp "C::f(3)"*** không cần đến đtượng cụ thể.
  - Không có khái niệm ~~this~~ trong pthức tĩnh!
    - Chỉ có thể truy cập đến các thuộc tính tĩnh của lớp.
  - Có thể truy cập đến những thành phần private của một đối tượng trong lớp (nhận qua tham số).

```
class C {
private:
   constexpr static bool debug = true;
   int i = 100; //non-static
public:
   static void testStatic(C &obj) {
     if (debug) { //static attribute, OK!
       //cout << i; //non-static attr, ERROR!
       cout << obj.i; //private attr of an instance, OK!
} } };
```

BT Ứng dụng 1: Số nguyên

- Đối tượng "số nguyên" có 1 thuộc tính là "giá trị" của nó, và các phương thức sau:
  - Khởi tạo mặc định (có giá trị 0), và khởi tạo có tham số (giá trị).
  - Đặt giá trị (gán giá trị tham số cho giá trị của số nguyên này).
  - Lấy giá trị (trả về giá trị của số nguyên này).
  - Cộng/trừ/nhân/chia với một số nguyên khác và trả về một số nguyên kết quả. (chia lấy phần nguyên)
    (Mỗi phép toán 1 phương thức riêng.)
  - Tăng/giảm số nguyên 1 đơn vị.
    (Mỗi phép toán 1 phương thức riêng.)
  - So sánh với một số nguyên khác và trả về "số âm nếu bé hơn, 0 nếu bằng, số dương nếu lớn hơn" số nguyên đó.
- Lớp số nguyên có các phương thức tĩnh sau:
  - Đếm số lượng các thực thể (đối tượng cụ thể) của lớp, trả về int.
  - Lấy "giá trị lỗi" khi chia cho 0 ("giá trị lỗi" là một hằng số của lớp).

BT Ứng dụng 1: Số nguyên (tiếp)

1. Hãy khai báo lớp "số nguyên" như đặc tả trên
2. Hãy viết hàm main() cho phép nhập 2 ***đối tượng*** số nguyên a, b; tính và xuất ra kết quả **cộng, trừ, nhân** 2 số nguyên đó; nếu **b khác 0** thì tính a **chia** b và xuất kết quả; **Tăng** a, **giảm** b 1 đơn vị rồi xuất ra giá trị của 2 số (sau khi tăng, giảm).
3. Cài đặt các phương thức của lớp "số nguyên".

BT ƯD 2: Mảng Số nguyên (Về nhà)

- Đối tượng "mảng số nguyên" có 2 thuộc tính là mảng chứa các đối tượng Số Nguyên và số phần tử hiện có trong mảng. Các phương thức gồm:
  - **Khởi tạo** mặc định: Mảng tĩnh (dung lượng cố định), chứa 0 p.tử.
  - **size**: Trả về số phần tử hiện có trong mảng.
  - **capacity**: Trả về dung lượng của mảng.
  - **empty**: Trả về true nếu mảng rỗng (chứa 0 phần tử).
  - **getAt**: Lấy (trả về) phần tử tại 1 vị trí trong mảng.
  - **setAt**: Đặt (gán) giá trị cho phần tử tại 1 vị trí trong mảng.
  - **push_back/pop_back**: Thêm/bớt 1 phần tử tại vị trí cuối mảng.
  - **insert/erase**: Chèn/xoá 1 phần tử tại 1 vị trí trong mảng.
  - **clear**: Xoá hết mảng (thành mảng rỗng).
  - **sum**: Trả về 1 đối tượng Số Nguyên là tổng của cả mảng.

Tóm tắt: Giao diện, Sử dụng

- Các cấp phụ thuộc: Lớp → Hàm → Biến
  - Khai báo (giao diện) & sử dụng phải tương thích với nhau.

	Khai báo (giao diện)	Sử dụng
Biến	Kiểu, tên biến	**Đọc/ghi giá trị** có kiểu tương ứng: • Khi đọc phải có giá trị xác định. **Truyền tham số** cho hàm: • Gián tiếp đọc/ghi giá trị.
Hàm	Nguyên mẫu hàm: • Kiểu trả về, tên hàm • Danh sách tham số	**Gọi hàm**: • Giá trị vào/ra phải có kiểu tương ứng với nguyên mẫu hàm.
Lớp	Giao diện: • Tên lớp • Các thành phần public Cấu trúc phần cài đặt: • Các thành phần private	**Nói chuyện** với đối tượng: • Gọi các phương thức public. • (Truy xuất các thuộc tính public) **Truyền tham số** cho hàm: • Gián tiếp sử dụng đối tượng.

Bảng đối chiếu thuật ngữ

Tiếng Việt	Tiếng Anh	Chú thích
Đối tượng	Object	
Lớp đối tượng (tắt "lớp")	Object class (abr. "class")	Là ***kiểu*** *của đối tượng*, chứ không phải là một tập hợp các đối tượng.
Thuộc tính	Attribute	Thành phần dữ liệu của đ.tượng, định nghĩa ***kiểu*** *d.liệu*
Phương thức	Method	Thành phần hành động của đ.tượng, định nghĩa ***cách thực hiện*** *hành động*.
Tái sử dụng	Reuse	
Giao diện (lập trình)	(Programming) interface	
Đóng gói / bao bọc	Encapsulation	
Phần cài đặt	Implementation	
Lập trình hướng đối tượng	Object oriented programming	

Bảng đối chiếu thuật ngữ (tt)

Tiếng Việt	Tiếng Anh	Chú thích
Kiểu dữ liệu (tắt "Kiểu")	Data type (abr. "Type")	
Kiểu cơ bản	Primitive type	
Kiểu phức hợp	Composite type	
Bộ nhớ dữ liệu	Data segment***s***	Phần bộ nhớ chương trình trừ Code segment
Thực thể, thực thể hoá	Instance, Instantiate	Còn gọi: Thể hiện, đối tượng (cụ thể)
Trạng thái / dữ liệu (của đối tượng)	State (of an object)	Là tập hợp các *giá trị cụ thể* của ***các thuộc tính*** trong một đ.tượng cụ thể.
Hành động (của đối tượng)	Action (of an object)	Là *thể hiện cụ thể* của ***phương thức*** (của lớp tương ứng) trên từng đối tượng.
Hành vi (của đối tượng)	Behavior (of an object)	Là *tập hợp các hành động* của đối tượng.

Phụ lục 1:
code khai báo lớp SinhVien

```
#pragma once

class SinhVien
{
private:
    float dLyThuyet; //điểm lý thuyết
    float dThucHanh; //điểm thực hành
public:
    SinhVien(); //khởi tạo các thuộc tính (điểm)
                //bằng giá trị mặc định (-1).
    void thiLT(); //(giáo viên) nhập điểm LT.
    void thiTH(); //(giáo viên) nhập điểm TH.
    float tinhDTK(); //trả về điểm tổng kết
private:
    float nhapDiem(string loaiDiem);
    //cho nhập điểm và đảm bảo hợp lệ (0 tới 10)
};
```

Phụ lục 2:
code cài đặt lớp SinhVien

```
#include "SinhVien.h"
#include<string>
#include <iostream>
using namespace std;

SinhVien::SinhVien()
   : dLyThuyet(-1), dThucHanh(-1)
     // -1 nghĩa là "chưa thi"
{
     // Không làm gì thêm hết.
}
```

Phụ lục 2:
code cài đặt lớp SinhVien (tiếp)

```
float SinhVien::nhapDiem(string loaiDiem)
{
     float diem;
     cout << loaiDiem<<": "; cin >> diem;
     if(diem > 10){ diem = 10; }
     if(diem < 0){ diem = 0; }
     return diem;
}
void SinhVien::thiLT()
{    this->dLyThuyet = this->nhapDiem("diem Ly thuyet"); }
void SinhVien::thiTH()
{    this->dThucHanh = this->nhapDiem("diem Thuc hanh"); }

float SinhVien::tinhDTK()
{
     float dtk = (this->dLyThuyet*6 + this->dThucHanh*4)/10;
     return dtk;
}
```