Khởi động trong Objective C

- Ta có thể khởi động đối tượng trong Objective C bằng cách định nghĩa lại phương thức `init` của lớp cơ sở NSObject.

```
-(instancetype)init {
    self = [super init];
    if (self) {
       numerator = 0;
       denominator = 1;// or [self set: 0 over: 1];
    }
    return self;
}
```

Tự khởi động 0/1
nếu quên khởi động

- Khởi động sẽ được nhắc lại ở chương sau.

```
int main(int argc, const char * argv[]) {
@autoreleasepool {
      Fraction *a = [[Fraction alloc] init];
      [a set : 1 over : 3];
      Fraction *b = [[Fraction alloc] init];
      [b set : 5 over : 6];
      Fraction *c = [[Fraction alloc] init];
      [a print];
      [b print];
      [c print];
   }
   return 0;
}
```

Khởi động tự động,
dọn dẹp tự động

```
void XuatHe16(long n)
{
   static char hTab[] = "0123456789ABCDEF";
   Stack s;
   int x;
   do {
      s.push(n % 16);
      n /= 16;
   } while (n);
   while (s.pop(&x))
   cout << hTab[x];
}
```

Trình tự sử dụng: Tạo đối tượng và sử dụng thực sự

Các đặc tính khác của lớp

- Tự tham chiếu
- Thuộc tính lớp và phương thức lớp (thành phần tĩnh)
- Hàm thành phần hằng (C++)
- Hàm bạn (C++)

Tự tham chiếu

- Là tham số ngầm định của phương thức trỏ đến đối tượng gọi thao tác. Nhờ đó phương thức biết được nó đang thao tác trên đối tượng nào.
- Khi một đối tượng gọi một thao tác, địa chỉ của đối tượng được gởi đi một cách ngầm định, trong C++ với tên this, trong Objective C là self.
- Tên các thành phần của đối tượng trong một phương thức được hiểu là của đối tượng có địa chỉ this hoặc self.

Con trỏ this

```
bool Stack::push(Item x)
{
   if (full()) // if (this->full())
      return false;
   *top++ = x; // *this->top++ = x;
   return true;
}
```

- Ta thường không cần tường minh sử dụng từ khoá this.

Từ khoá self

```
-(Fraction *)addIn: (Fraction *)b {
    numerator = numerator * b.denominator +
    denominator * b.numerator;
    denominator *= b.denominator;
    [self reduce]; // use self to call reduce
    return self;
}
```

Hoặc

```
-(Fraction *)addIn: (Fraction *)b {
    // use self to call set
    [self set: numerator * b.denominator +
       denominator * b.numerator over:
       denominator * b.denominator];
    return self;
}
```

Phương thức hằng

```
inline char *_strdup(const char *s) {
    return strcpy(new char[strlen(s) + 1], s);
}

class String {
    char *p;
public:
    String(const char *s) { p = _strdup(s); }
    String(const String &s) { p = _strdup(s.p); }
    ~String() { if (p) delete p; }
    int getLength() { return strlen(p); }
    void print();
    void set(const char *s);
    void input();
    void concat(const String &b);
};
```

```
int main()
{
    String b("This is a string");
    b.print();
    cout << "\n";
    b.set("a variable string");
    b.print();
    cout << "\n";

    String mySchool("Dai Hoc Tu Nhien");
    mySchool.print();
    cout << "\n";
    mySchool.set("Tieu Hoc Tu Nhien");
    mySchool.print();
    cout << "\n";

    return 0;
}
```

```
int main()
{
    String b("This is a string");
    b.print();
    cout << "\n";
    b.set("a variable string");
    b.print();
    cout << "\n";

    const String mySchool("Dai Hoc Tu Nhien");
    mySchool.print();
    cout << "\n";
    mySchool.set("Tieu Hoc Tu Nhien");
    mySchool.print();
    cout << "\n";

    return 0;
}
```

Phương thức hằng

- Phương thức hằng hay hàm thành phần hằng là phương thức có thể áp dụng được cho các đối tượng hằng.
- Ta qui định một phương thức là hằng bằng cách thêm từ khoá const vào cuối khai báo của nó.
- Ta khai báo phương thức là hằng khi nó không thay đổi các thành phần dữ liệu của đối tượng.

Phương thức hằng

```
inline char *_strdup(const char *s) {
    return strcpy(new char[strlen(s) + 1], s);
}

class String {
    char *p;
public:
    String(const char *s) { p = _strdup(s); }
    String(const String &s) { p = _strdup(s.p); }
    ~String() { if (p) delete p; }
    int getLength() const { return strlen(p); }
    void print() const;
    void set(const char *s);
    void input();
    void concat(const String &b);
};
```

```
int main()
{
    String b("This is a string");
    b.print();
    cout << "\n";
    b.set("a variable string");
    b.print();
    cout << "\n";

    const String mySchool("Dai Hoc Tu Nhien");
    mySchool.print();
    cout << "\n";
    mySchool.set("Tieu Hoc Tu Nhien");
    mySchool.print();
    cout << "\n";

    return 0;
}
```

Hàm bạn

- Nguyên tắc chung khi thao tác trên lớp là thông qua các hàm thành phần (setters, getters). Tuy nhiên có những trường hợp ngoại lệ, khi hàm phải thao tác trên hai lớp.
- Hàm bạn của một lớp là hàm được khai báo ở bên ngoài nhưng được phép truy xuất các thành phần riêng tư của lớp.

Hàm bạn

- Ta làm một hàm trở thành hàm bạn của lớp bằng cách đưa khai báo của hàm đó vào trong lớp, thêm từ khoá friend ở đầu.
- Ta dùng hàm bạn trong trường hợp hàm phải là hàm toàn cục nhưng có liên quan mật thiết với lớp, hoặc là hàm thành phần của một lớp khác.

Nhân ma trận, không dùng hàm bạn

```
const int N = 4;
class Vector
{
   double a[N];
   public:
   double Get(int i) const { return a[i]; }
   void Set(int i, double x) { a[i] = x; }
};

class Matrix
{
   double a[N][N];
   public:
   double Get(int i, int j) const { return a[i][j]; }
   void Set(int i, int j, double x) { a[i][j] = x; }
   //...
};
```

Nhân ma trận, không dùng hàm bạn

```
Vector Multiply(const Matrix &m, const
Vector &v)
{
   Vector r;
   for (int i = 0; i < N; i++)
   {
      r.Set(i, 0);
      for (int j = 0; j < N; j++)
         r.Set(i, r.Get(i) + m.Get(i,
         j)*v.Get(j));
   }
   return r;
}
```

Nhân ma trận, dùng hàm bạn

```
class Matrix;
class Vector
{
   double a[N];
   public:
   double Get(int i) const { return a[i]; }
   void Set(int i, double x) { a[i] = x; }
   friend Vector Multiply(const Matrix &m, const Vector &v);
};

class Matrix
{
  double a[N][N];
  public:
  double Get(int i, int j) const { return a[i][j]; }
  void Set(int i, int j, double x) { a[i][j] = x; }
  friend Vector Multiply(const Matrix &m, const Vector &v);
};
```

Nhân ma trận, dùng hàm bạn

```
Vector Multiply(const Matrix &m, const
Vector &v)
{
   Vector r;
   for (int i = 0; i < N; i++)
   {
      r.a[i] = 0;
      for (int j = 0; j < N; j++)
      r.a[i] += m.a[i][j] * v.a[j];
   }
   return r;
}
```

Thuộc tính lớp và phương thức lớp

- Thuộc tính lớp là thuộc tính dùng chung cho mọi đối tượng thuộc lớp.
- Phương thức lớp là phương thức có thể hoạt động không thuộc tính dữ liệu của lớp, nghĩa là nó không cần đối tượng.
- Ta dùng phương thức lớp thay cho hàm toàn cục.
- Trong C++, từ khoá static được dùng để khai báo thuộc tính hay phương thức là của lớp.

```
class CDate
{
   static int dayTab[][13];
   int day, month, year;
public:
   static bool LeapYear(int y) { return y %
   400 == 0 || y % 4 == 0 && y % 100 != 0; }
   static int DayOfMonth(int m, int y);
   static bool ValidDate(int d, int m, int y);
   void Input();
};
```

```
class CDate
{
   static int dayTab[][13];
   int day, month, year;
public:
   static bool LeapYear(int y) { return y %
   400 == 0 || y % 4 == 0 && y % 100 != 0; }
   static int DayOfMonth(int m, int y);
   static bool ValidDate(int d, int m, int y);
   void Input();
};
```

```
int CDate::dayTab[][13] =
{
   { 0,31,28,31,30,31,30,31,31,30,31,30,31 },
   { 0,31,29,31,30,31,30,31,31,30,31,30,31 }
};

int CDate::DayOfMonth(int m, int y)
{
   return dayTab[LeapYear(y)][m];
}

bool betw(int x, int a, int b)
{
   return x >= a && x <= b;
}
```

```
bool CDate::ValidDate(int d, int m, int y)
{
   return betw(m, 1, 12) && betw(d, 1, DayOfMonth(m,
   y));
}
void CDate::Input()
{
   int d, m, y;
   cin >> d >> m >> y;
   while (!ValidDate(d, m, y))
   {
      cout << "Please enter a valid date: ";
      cin >> d >> m >> y;
   }
   day = d; month = m; year = y;
}
```

Thuộc tính và phương thức lớp

```
class Stack
{
    Item *st, *top;
    int size;
    void init(int sz);
    void cleanUp() { if (st) delete[] st; }
public:
    Stack(int sz = 20) { init(sz); }
    ~Stack() { CleanUp(); }
    static Stack *create(int sz);
    bool full()const { return (top - st >= size); }
    bool empty() const { return (top <= st); }
    bool push(Item x);
    bool pop(Item *px);
};
```

```
Stack *Stack::create(int sz)
{
    Stack *ps = new Stack(sz);
    if (!ps->st)
    {
        delete ps;
        return NULL;
    }
    return ps;
}
void main()
{
    Stack *ps = new Stack(50000); // ko biet tao duoc stack khong
    Stack *pr = Stack::create(50000);
    if (!pr)
        { cout << "Khong the tao stack"; return; }
    else
        pr->push(5); // ...
}
```