2.2. CÁC CẤU TRÚC ĐIỀU KHIỂN

- CẤU TRÚC RẼ NHÁNH
- CẤU TRÚC LẶP
- PHƯƠNG THỨC
- XỬ LÝ NGOẠI LỆ

1. CẤU TRÚC RẼ NHÁNH

- Cấu trúc if
- Cấu trúc switch . . . case

CẤU TRÚC IF

Câu lệnh if đơn giản

- Cú pháp

```
if (biểu thức điều kiện)
{
    //Khối lệnh A
}
```

- Ý nghĩa:
  Nếu ***biểu thức điều kiện*** đúng thì thực hiện Khối lệnh A

CẤU TRÚC IF

Câu lệnh if …else

Cú pháp

```
if (biểu thức điều kiện)
{
    //Khối lệnh A
}
[else
{
    //Khối lệnh B
}]
```

Ý nghĩa:
Nếu *biểu thức điều kiện* đúng thì
thực hiện Khối lệnh A
Ngược lại, *biểu thức điều kiện* sai
thực hiện Khối lệnh B

CẤU TRÚC IF

- **Chú ý:**
  - Nếu khối lệnh chỉ có 1 câu lệnh thì không cần dùng cặp ngoặc nhọn { }
  - Nếu có một chuỗi các điều kiện xử lý liên tục, có thể sử dụng nhiều if lồng nhau.

CẤU TRÚC IF

Câu lệnh if …else if

Cú pháp

```
if (biểu thức điều kiện 1)
{
    //Khối lệnh 1
}
else if (biểu thức điều kiện 2)
{
    //Khối lệnh 2
}
.....
else
{
    //Khối lệnh n
}
```

Ý nghĩa:
Nếu *biểu thức điều kiện 1* đúng thì
thực hiện Khối lệnh 1
Ngược lại, *biểu thức điều kiện 2 đúng thì*
thực hiện Khối lệnh 2
…..
Ngược lại tất cả các điều kiện trên thì
thực hiện Khối lệnh n

CẤU TRÚC SWITCH

- **Cú pháp**

```
switch(biểu thức điều khiển)
{
    case giá trị 1:
        //Tập lệnh 1
        break;
    case giá trị 2:
        //Tập lệnh 2
        break;
    …
    default:
        //Tập lệnh n
        break;
}
```

Ý nghĩa

- **case**: liệt kê các trường hợp cần xét
- **giá trị i**: là các giá trị hằng cần so sánh với biểu thức điều khiển
- Nếu biểu thức điều khiển bằng giá trị i thì tập lệnh i được thực hiện.
- Nếu các case không thỏa thì thực hiện phát biểu **default** (nếu có)

CẤU TRÚC SWITCH

- Các giá trị của các phát biểu case phải khác nhau, không có 2 case có cùng giá trị
- Kết thúc phát biểu case phải có break.
- Nếu muốn nhiều case cùng thực hiện một tập lệnh thì viết như sau:

```
. . .
case giá trị 1:
case giá trị 2:
. . .
//Tập lệnh chung
```


CẤU TRÚC SWITCH

Chú ý:

- Chỉ sử dụng switch cho các kiểu dữ liệu nguyên thủy như: int, string, char, bool… (biến thuộc kiểu dữ liệu hữu hạn, đếm được)những kiểu khác kể cả float, double nên sử dụng if
- Các giá trị phải là một hằng số như: 10, “Hà nội”. . . Nếu cần tính toán dữ liệu so sánh khi runtime thì nên dùng hàm if

3.2. CẤU TRÚC LẶP

- while
- do…while
- for
- break và continue

CẤU TRÚC WHILE

- Thực thi vòng lặp nhiều lần nhưng không biết trước sẽ lặp bao nhiêu lần trước khi vòng lặp thực hiện
- **Cú pháp**

```
while (<biểu_thức_logic>)
{
    // Khối lệnh
    . . .
}
```

Ý nghĩa: Kiểm tra biểu_thức_logic, nếu biểu_thức_logic được đánh giá là true thì thực hiện khối lệnh. Tiếp tục thực hiện khối lệnh cho đến khi biểu_thức_logic là false

CẤU TRÚC WHILE

```
. . .
int n = 1;
while (n < 6)
{
    Console.WriteLine(“Gia tri cua n la: {0}", n);
    n++;
}
/* Kết quả
   Gia tri cua n la: 1
   Gia tri cua n la: 2
   Gia tri cua n la: 3
   Gia tri cua n la: 4
   Gia tri cua n la: 5
*/
```

CẤU TRÚC DO . . . WHILE

- **Cú pháp**

```
do
{
    // khối lệnh
    . . .
} while (<biểu_thức_logic);
```

Ý nghĩa:
khối lệnh trong vòng lặp sẽ được thực thi trước, sau đó kiểm tra biểu_thức_logic. Nếu biểu_thức_logic được đánh giá là true thì khối lệnh tiếp tục được thực hiện, nếu là false thì vòng lặp kết thúc

CẤU TRÚC DO . . . WHILE

```
. . .
int x = 10;
do
{
   Console.WriteLine(x);
   x++;
 } while (x < 5);
```

```
/*
   Kết quả:
   0
   1
   2
   3
   4
*/
```


CẤU TRÚC FOR

- Sử dụng khi đã biết trước số lần lặp

Cú pháp

```
for (biểu_thức_khởi_tạo; biểu_thức_boolean;
biểu_thức_tăng)
{
     //khối lệnh
}
```

- **biểu_thức_khởi_tạo**: khai báo và gán giá trị ban đầu cho một biến đếm
- **biểu_thức_boolean**: xác định điều kiện dừng của vòng lặp
- **biểu_thức_tăng**: tăng hoặc giảm giá trị của biến đếm sau mỗi lần lặp

CẤU TRÚC FOR

```
. . .
for (int i = 1; i <= 5; i++)
{
 Console.WriteLine(i);
}
```

```
/*
   Kết quả:
   1
   2
   3
   4
   5
*/
```


CẤU TRÚC LẶP

- **VD1**: viết chương trình cho phép người dùng nhập vào số tiền tiết kiệm hàng tháng, lãi suất tháng. Tính và hiển thị số tiền thu được hàng tháng. Chương trình **kết thúc khi bạn trở thành triệu phú**.
- **VD2**: viết chương trình cho phép người dùng nhập vào số tiền tiết kiệm hàng tháng, lãi suất tháng, số tháng gửi tiết kiệm. Tính và hiển thị số tiền thu được hàng tháng.

SỬ DỤNG BREAK & CONTINUE TRONG VÒNG LẶP

- Các vòng lặp kết thúc khi biểu thức boolean có giá trị false
- Các lệnh jump cho phép điều khiển thứ tự thực thi các lệnh
  - break: kết thúc vòng lặp (thoát khỏi vòng lặp)
  - continue: chuyển đến lần lặp tiếp theo

3. SỬ DỤNG PHƯƠNG THỨC

- **Phương thức** (method) là một chuỗi các câu lệnh được đặt tên, nhằm thực hiện các tác vụ cụ thể nào đó.
  - Có thể có tham số hoặc không
  - Có thể trả về hoặc không trả về giá trị
  - Được thực thi bằng cách gọi tên phương thức và cung cấp các đối số cần thiết

3. SỬ DỤNG PHƯƠNG THỨC

Cú pháp định nghĩa phương thức:

```
điều_khiển_truy_cập
kiểu_giá_trị_trả_về   tên_phương_thức
([danh_sách_tham_số])
{
    // thân phương thức
}
```


3. SỬ DỤNG PHƯƠNG THỨC

- **điều_khiển_truy_cập**: quy định cách thức truy cập phương thức
  - private: phương thức chỉ có thể được gọi từ một phương thức khác trong cùng lớp
  - public: phương thức được gọi từ bên ngoài lớp

3. SỬ DỤNG PHƯƠNG THỨC

- **kiểu_giá_trị_trả_về**: chỉ ra kiểu dữ liệu của giá trị trả về.
  - Kiểu dữ liệu trả về có thể là kiểu dữ liệu nguyên thủy, class, struct, enum . . .
  - Nếu phương thức không trả về giá trị thì sử dụng từ khóa void

3. SỬ DỤNG PHƯƠNG THỨC

- **tên_phương_thức**: theo quy tắc định danh
- **danh_sách_tham_số**: được sử dụng để truyền và nhận dữ liệu từ phương thức
- **thân phương thức**: là một tập các lệnh cần thiết để hoàn thành tác vụ yêu cầu
- Tên và danh sách tham số tạo nên signature (dấu hiệu) của phương thức

3. SỬ DỤNG PHƯƠNG THỨC

Gọi phương thức:

```
tên_phương_thức ([danh_sách_đối_số])
```

3. SỬ DỤNG PHƯƠNG THỨC

Ví dụ:

```
. . .
//định nghĩa phương thức
 public static int Cong(int so1, int so2)
 {
       return so1+ so2;
 }
. . .
//gọi phương thức
 kq = Cong(2, 4);
. . .
```


3. SỬ DỤNG PHƯƠNG THỨC

Trả về giá trị cho phương thức

- Phương thức có thể trả về **một** giá trị
- Sử dụng câu lệnh return để trả về giá trị cho phương thức.

  return giá_trị_trả_về;

  – return; //dừng thực thi phương thức
  – Nếu không có từ khóa return, phương thức dừng thực thi khi đến cuối khối lệnh

3. SỬ DỤNG PHƯƠNG THỨC

Tham số và đối số

- Định nghĩa phương thức xác định tên và kiểu của các tham số (parameter).
- Khi gọi phương thức, ta cung cấp các giá trị cụ thể gọi là đối số (argument) cho mỗi tham số.
- Các đối số phải tương thích với các kiểu của tham số
- Tên đối số nếu được sử dụng trong lời gọi không phải giống như tên các tham số được định nghĩa trong phương thức.

3. SỬ DỤNG PHƯƠNG THỨC

Ví dụ:

```
int Square(int i)
{
  //lưu tham số vào biến cục bộ
  int input = i;
  return input * input;
}
```

```
public void Caller()
{
    int numA = 4;
    //gọi với một biến int
    int productA = Square(numA);
    int numB = 32;
    //gọi với biến int khác
    int productB = Square(numB);
    //gọi với một hằng số int
    int productC = Square(12);
    //gọi với một biểu thức đánh giá là int
    productC = Square(productA * 3);
}
```

3. SỬ DỤNG PHƯƠNG THỨC

Truyền tham số cho phương thức

- Các tham số của phương thức có thể được truyền theo các cách sau:
  – Value (giá trị):
  – Reference(tham chiếu):
  – Output
- **Cú pháp khai báo tham số :**

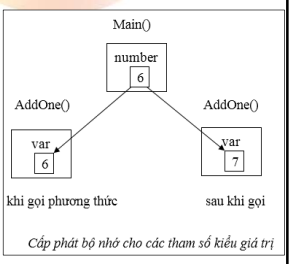
  [ref | out] <kiểu_dữ_liệu> tên_tham_số

3. SỬ DỤNG PHƯƠNG THỨC

Truyền tham số theo giá trị (mặc định)

- Ví dụ:

```
static void Main(string[] args)
{
     int number = 6;
     AddOne(number);
     Console.WriteLine(number);
}
static void AddOne(int var)
{
     var++;
}
```



3. SỬ DỤNG PHƯƠNG THỨC

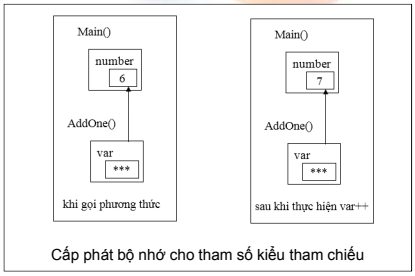
Truyền tham số theo tham chiếu

- Ví dụ:

```
static void Main(string[] args)
{
      int number = 6;
      AddOne(ref number);
      Console.WriteLine(number);
}
static void AddOne(ref int var)
{
      var++;
}
```


3. SỬ DỤNG PHƯƠNG THỨC

Truyền tham số theo tham chiếu



Cấp phát bộ nhớ cho tham số kiểu tham chiếu

3. SỬ DỤNG PHƯƠNG THỨC

Truyền tham số output

- Tương tự tham số tham chiếu nhưng tham số output chỉ truyền dữ liệu từ phương thức ra, không nhận dữ liệu vào
- Không cần gán giá trị cho biến trước khi gọi tham số
- Tham số output phải được gán giá trị trong thân phương thức
- Phương thức chỉ trả về 1 giá trị → sử dụng tham số output để trả về nhiều giá trị

3. SỬ DỤNG PHƯƠNG THỨC

Tham số tùy chọn

- Cung cấp giá trị mặc định cho 1 hoặc nhiều tham số.
- Khi gọi phương thức, nếu người dùng không cung cấp đối số, giá trị mặc định sẽ được sử dụng
- Khai báo tham số tùy chọn

kiểu_dữ_liệu tên_tham_số = giá_trị_mặc_định

3. Sử dụng phương thức

Tham số có số lượng đối số khác nhau

- Từ khóa **params** cho phép chỉ định một tham số của phương thức có số lượng đối số thay đổi
- Bạn có thể truyền 0 đối số, 1 danh sách đối số có kiểu được quy định trong khai báo của tham số - mỗi đối số cách nhau một dấu phẩy hoặc một mảng chứa các tham số có kiểu quy định
- Trong khai báo phương thức chỉ có 1 từ khóa params và không được có thêm tham số nào sau từ khóa params

4. Xử lý ngoại lệ và kiểm tra dữ liệu hợp lệ

- ❖ Chạy và gỡ lỗi ứng dụng
- ❖ Xử lý ngoại lệ
- ❖ Kiểm tra dữ liệu hợp lệ

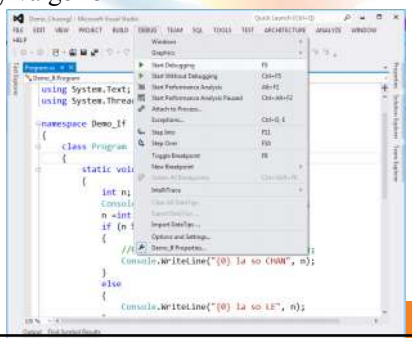
3.4.1. Chạy và gỡ lỗi ứng dụng

Các loại lỗi:

- Lỗi cú pháp (syntax error)
- Lỗi runtime
- Lỗi logic

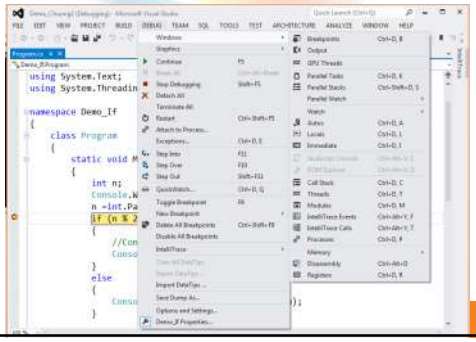
3.4.1. Chạy và gỡ lỗi ứng dụng

Chạy và gỡ lỗi



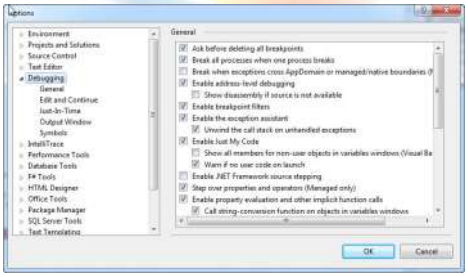
3.4.1. Chạy và gỡ lỗi ứng dụng

Các tính năng gỡ lỗi



3.4.1. Chạy và gỡ lỗi ứng dụng

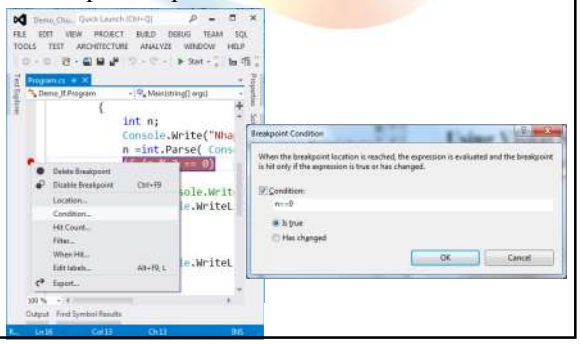
Các tính năng gỡ lỗi



Cấu hình bắt lỗi trong visual studio

3.4.1. Chạy và gỡ lỗi ứng dụng

Thiết lập Breakpoint có điều kiện



3.4.2. Xử lý ngoại lệ

Ngoại lệ là gì?

- Ngoại lệ là vấn đề phát sinh trong quá trình thực hiện chương trình (runtime error).
- Ngoại lệ thường là một điều kiện lỗi hoặc sự kiện khác làm gián đoạn luồng thực thi bình thường của ứng dụng.

3.4.2. Xử lý ngoại lệ

Tại sao phải xử lý ngoại lệ?

- Trong thực tế chúng ta không muốn chương trình dừng lại một cách bất thường

  ➔ cần phải kiểm soát và xử lý ngoại lệ

3.4.2. Xử lý ngoại lệ

Mục đích của xử lý ngoại lệ:

- Cho phép chương trình tiếp tục thực hiện mà không bị dừng đột ngột khi gặp ngoại lệ
- Thông báo cho người sử dụng vấn đề mà chương trình gặp phải một cách có ý nghĩa
- Kết thúc chương trình một cách có kiểm soát

3.4.2. Xử lý ngoại lệ

Cú pháp

```
try
{
    //Các câu lệnh có thể gây ra ngoại lệ
}
catch
{
    //Các câu lệnh để xử lý ngoại lệ
}
finally
{
    //Các lệnh thực thi dù có ngoại lệ hay không
}
```


3.4.2. Xử lý ngoại lệ

```
try
{
    //Các câu lệnh có thể phát sinh ngoại lệ
}
catch (tên_ngoại_lệ e1)
{
    //code xử lý ngoại lệ
}
. . .
catch (tên_ngoại_lệ  eN)
{
    //code xử lý ngoại lệ
}
finally
{
    //các lệnh thực thi dù có ngoại lệ hay không
}
```

Dùng nhiều khối catch để bắt các ngoại lệ khác nhau trong trường hợp khối try phát sinh nhiều ngoại lệ

3.4.2. Xử lý ngoại lệ

- Các ngoại lệ của C# được biểu diễn bởi các class
- Các ngoại lệ đều dẫn xuất từ lớp **System.Exception**
- Các lớp ngoại lệ
  - **System.ApplicationException:** class hỗ trợ các ngoại lệ phát sinh bởi các chương trình ứng dụng
  - **System.SystemException**: là lớp cơ sở cho tất cả các ngoại lệ hệ thống được định nghĩa trước như FormatExcetption, ArgumentExcetpion . . .

3.4.2. Xử lý ngoại lệ

Một số ngoại lệ thường gặp

Tên ngoại lệ	Mô tả
DivideByZeroException	chia cho 0
IndexOutOfRangeException	chỉ số truy cập mảng không hợp lệ
FormatException	định dạng không chính xác của một đối số nào đó
InvalidCastException	lỗi phát sinh khi ép kiểu
OutOfMemoryException	tràn bộ nhớ

Có nhiều lớp Exception khác trong C#, bạn hãy tìm hiểu về chúng!!

3.4.3. Kiểm tra hợp lệ dữ liệu

- Việc kiểm tra dữ liệu người dùng nhập vào đảm bảo tính hợp lệ gọi là kiểm tra hợp lệ dữ liệu (data validation)
- Khi dữ liệu đầu vào không hợp lệ chương trình cần hiển thị thông báo lỗi cho người dùng và có xử lý thích hợp
- Các kiểu kiểm tra hợp lệ: (1) kiểm tra đầu vào bắt buộc (2) kiểm tra đầu vào định dạng số (3) Kiểm tra giá trị nằm trong một khoảng xác định

QUY ƯỚC VIẾT CODE

- Biến cục bộ, tham số: ký pháp Camel – chữ thường sau đó viết hoa ký tự đầu
  - Ví dụ: newUser, inputParameter
- Trường hợp đặc biệt:
  - Chỉ số, biến đơn giản: chữ thường - i, j, name
  - Hằng: CHỮ HOA và dấu gạch dưới _: MAX_AGE
  - Biến Boolean chỉ trạng thái: isReady, isFinish
- Các trường hợp khác – Ký pháp Pascal : Viết hoa ký tự đầu mỗi từ
  - Ví dụ: AddUser, Color