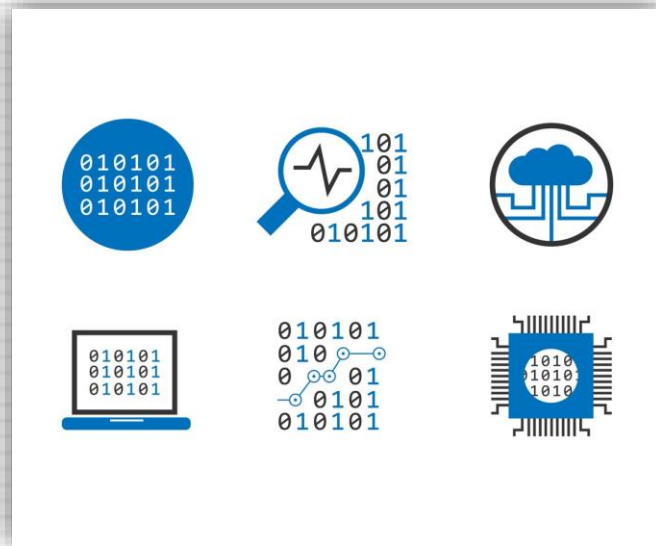


# KỸ THUẬT LẬP TRÌNH

## CHƯƠNG 1: CON TRỎ (POINTER)

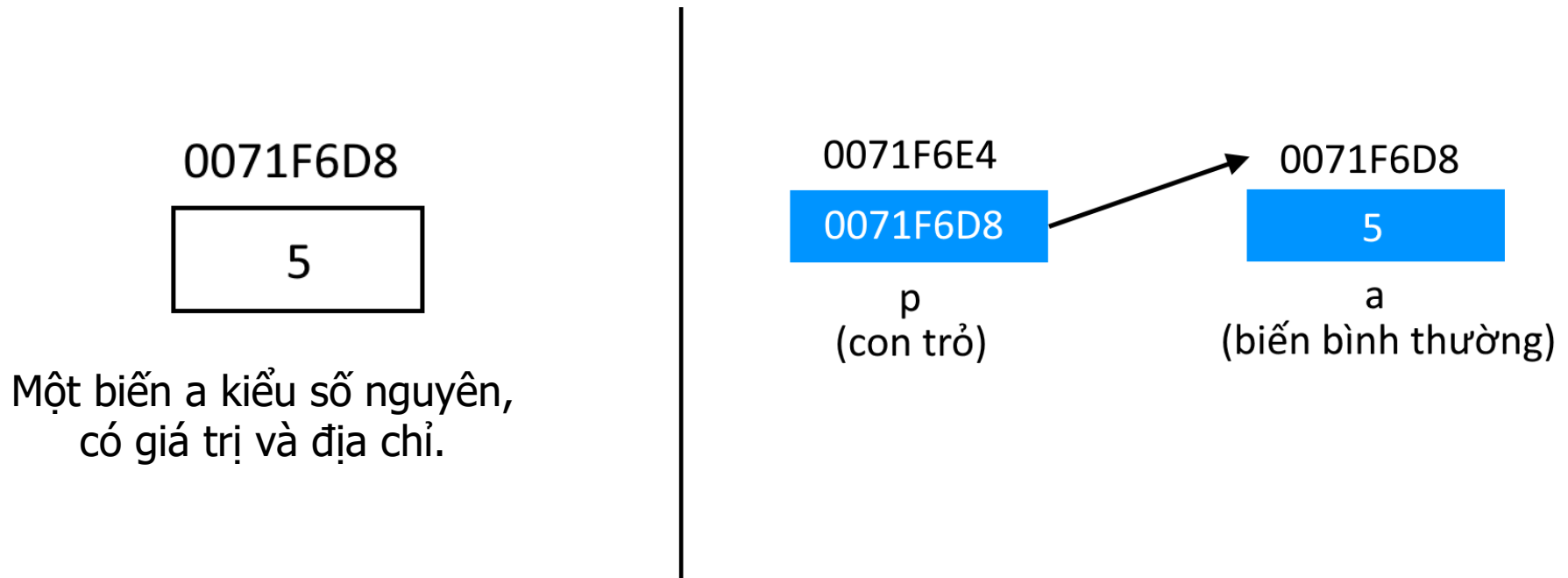


GV: Phạm Nguyễn Sơn Tùng

Email: [pnstung@fit.hcmus.edu.vn](mailto:pnstung@fit.hcmus.edu.vn)

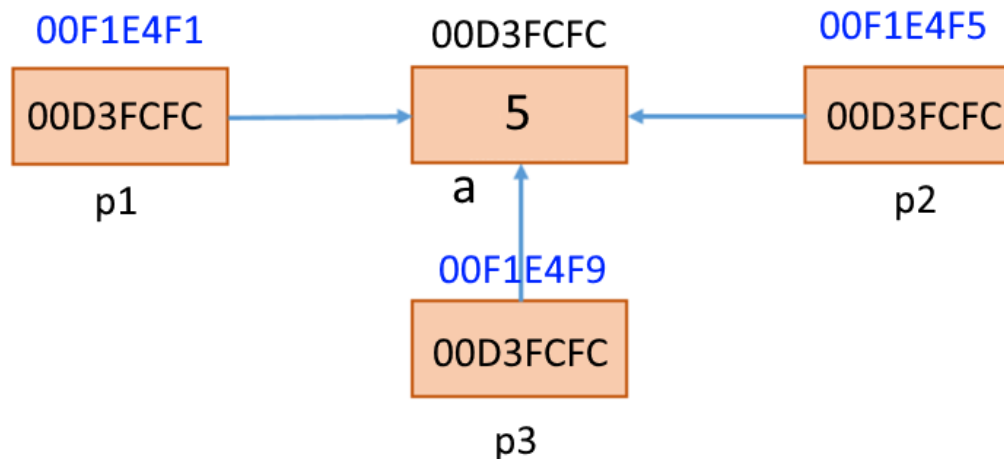
# GIỚI THIỆU VỀ CON TRỎ

**Định nghĩa:** con trỏ là một biến, nhưng biến con trỏ không lưu giá trị bình thường giống như các biến khác, nó lưu địa chỉ của một biến khác.



## TẠI SAO PHẢI SỬ DỤNG CON TRỎ

Về bản chất con trỏ cũng như một biến bình thường, có tên biến, có giá trị của biến, địa chỉ của biến. Nhưng khác các biến bình thường chỉ nằm cố định trong 1 ô nhớ, còn con trỏ có thể trỏ đến các ô nhớ khác nhau.



# CÚ PHÁP KHAI BÁO CON TRỎ

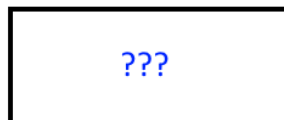
**Cú pháp:** <kiểu dữ liệu> \* <tên biến>

```
int main()
{
    int* p1;
    float* p2;
    return 0;
}
```

p1 là biến con trỏ, trỏ tới vùng nhớ kiểu số nguyên 4 byte.

p2 là biến con trỏ, trỏ tới vùng nhớ kiểu số thực 4 byte.

0071F6D8



p1

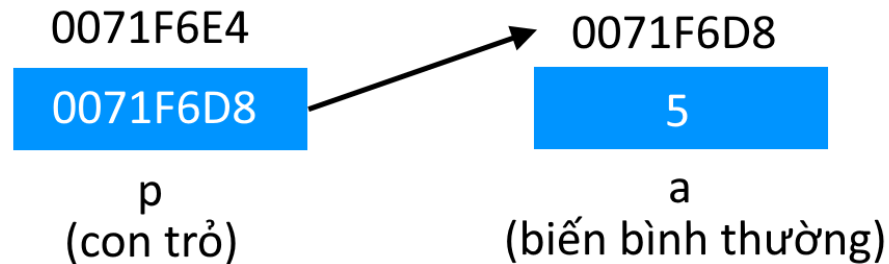
## CÚ PHÁP SỬ DỤNG CON TRỎ

Dấu **\*** dùng để khai báo con trỏ, cũng vừa để truy cập đến nội dung con trỏ đang trỏ đến.

Dấu **&** dùng để truy xuất đến vùng nhớ của con trỏ.

```
int main()
{
    int* p;
    int a = 5;
    p = &a;
    cout << "1. Địa chỉ của biến a: " << &a << endl;
    cout << "2. Giá trị của biến a: " << a << endl;
    cout << "3. Địa chỉ của con trỏ: " << &p << endl;
    cout << "4. Giá trị của con trỏ: " << p << endl;
    cout << "5. Nội dung của biến mà con trỏ, trỏ tới: " << *p;
    return 0;
}
```

# MỘT SỐ THAO TÁC VỚI CON TRỎ



```
int main()
{
    int* p;
    int a = 5;
    p = &a;
    cout << "1. Dia chi cua bien a: " << &a << endl;
    cout << "2. Gia tri cua bien a: " << a << endl;
    cout << "3. Dia chi cua con tro: " << &p << endl;
    cout << "4. Gia tri cua con tro: " << p << endl;
    cout << "5. Noi dung cua bien ma con tro, tro toi: " << *p;
    return 0;
}
```

# MỘT SỐ THAO TÁC VỚI CON TRỎ

2 mã nguồn sau có sự khác biệt gì hay không?

```
int main()
{
    int a = 5;
    int* p = &a;
    return 0;
}
```

```
int main()
{
    int a = 5;
    int* p;
    *p = &a;
    return 0;
}
```

# MỘT SỐ THAO TÁC VỚI CON TRỎ

## 1. Khai báo con trỏ tự cấp phát vùng nhớ và giá trị

```
int main()
{
    int* p = new int;
    *p = 5;
    cout << "1. Địa chỉ của con trỏ: " << &p << endl;
    cout << "2. Giá trị của con trỏ: " << p << endl;
    cout << "3. Nội dung của biến mà con trỏ, trỏ tới: " << *p;
    return 0;
}
```



# MỘT SỐ THAO TÁC VỚI CON TRỎ

## 2. Con trỏ gán cho con trỏ.

```
int main()
{
    int* p1 = new int;
    *p1 = 5;


    cout << "1. Địa chỉ của con trỏ p1: " << &p1 << endl;
    cout << "2. Giá trị của con trỏ p1: " << p1 << endl;
    cout << "3. Nội dung của biến mà con trỏ, trỏ tới: " << *p1 << endl << endl;

    int* p2 = p1;
    cout << "4. Địa chỉ của con trỏ p2: " << &p2 << endl;
    cout << "5. Giá trị của con trỏ p2: " << p2 << endl;
    cout << "6. Nội dung của biến mà con trỏ, trỏ tới: " << *p2;
    return 0;
}
```

# MỘT SỐ THAO TÁC VỚI CON TRỎ

## 3. Khi tăng nội dung của biến con trỏ đang trỏ tới lên 1.

```
int main()
{
    int* p = new int;
    *p = 5;
    int a = *p++;
    cout << "1. Địa chỉ của con trỏ p1: " << &p << endl;
    cout << "2. Giá trị của con trỏ p1: " << p << endl;
    cout << "3. Nội dung của biến mà con trỏ, trỏ tới: " << *p << endl << endl;
    cout << a << endl;
    cout << *p << endl;
    return 0;
}
```



Chuyện gì  
sẽ xảy ra?

# MỘT SỐ THAO TÁC VỚI CON TRỎ


## 3. Một số phép toán khác của con trỏ.

```
int main()
{
    int a = 5;
    int *p = &a;
    p++;
    p--;
    p = p + 2;
    p = p - 2;
}
```

# MỘT SỐ THAO TÁC VỚI CON TRỎ

## 4. Phép gán con trỏ, chỉ gán được các con trỏ cùng kiểu.

```
int main()
{
    int a = 5;
    int* p1 = &a;
    int* p2 = p1;
    return 0;
}
```



Nếu gán khác  
kiểu chuyện gì  
sẽ xảy ra?

## MỘT SỐ THAO TÁC VỚI CON TRỎ

5. Các toán tử so sánh `==`, `>`, `<`, `!=` nếu 2 con trỏ **khác kiểu dữ liệu** thì chỉ so sánh được giá trị mà con trỏ đang trỏ tới, các trường hợp khác sẽ báo lỗi.

```
int main()
{
    int a = 5, * p1 = &a;
    double b = 5, * p2 = &b;
    // so sánh 2 con trỏ khác kiểu
    if (*p1 == *p2) //( &p1 == &p2) //(p1 == p2)
        cout << "p1 bang p2";
    else
        cout << "p1 khong bang p2";
}
```

## MỘT SỐ THAO TÁC VỚI CON TRỎ

5. Nếu 2 con trỏ **cùng kiểu** thì bạn có thể so sánh tất cả mọi thứ của 2 con trỏ: địa chỉ, giá trị, nội dung trỏ đến.

```
int main()
{
    int a = 5, * p1 = &a;
    int b = 5, * p2 = &b;
    // so sánh 2 con trỏ cùng kiểu
    if (*p1 == *p2) //( &p1 == &p2) //(p1 == p2)
        cout << "p1 bang p2";
    else
        cout << "p1 khong bang p2";
}
```

## MỘT SỐ THAO TÁC VỚI CON TRỎ


5. Nếu 2 con trỏ **cùng kiểu** thì bạn có thể so sánh tất cả mọi thứ của 2 con trỏ: địa chỉ, giá trị, nội dung trỏ đến.

```
int main()
{
    int a = 5, * p1 = &a;
    int* p2 = &a;
    p1++;
    if (p1 >= p2) //(p1 == p2) //(p1 == p2)
        cout << "p1 lưu trữ sau p2";
    else
        cout << "p1 lưu trữ trước p2";
}
```

# TRUYỀN THAM SỐ CHO CON TRỎ

## 1. Truyền tham trị hoán đổi giá trị của 2 con trỏ.

```
void HoanVi(int p1, int p2)
{
    int temp = p1;
    p1 = p2;
    p2 = temp;
}
int main()
{
    int a = 5, * p1 = &a;
    int b = 7, * p2 = &b;
    HoanVi(*p1, *p2);
    cout << "p1: " << *p1 << endl;
    cout << "p2: " << *p2 << endl;
}
```



Chuyện gì  
sẽ xảy ra?



# TRUYỀN THAM SỐ CHO CON TRỎ

## 2. Truyền tham chiếu hoán đổi giá trị của 2 con trỏ.

```
void HoanVi(int &p1, int &p2)
{
    int temp = p1;
    p1 = p2;
    p2 = temp;
}
int main()
{
    int a = 5, * p1 = &a;
    int b = 7, * p2 = &b;
    HoanVi(*p1, *p2);
    cout << "p1: " << *p1 << endl;
    cout << "p2: " << *p2 << endl;
}
```

# TRUYỀN THAM SỐ CHO CON TRỎ

## 3. Nếu tham số truyền vào là 2 con trỏ thì sao?

```
void HoanVi(int *p1, int *p2)
{
    int *temp = p1;
    p1 = p2;
    p2 = temp;
}
int main()
{
    int a = 5, * p1 = &a;
    int b = 7, * p2 = &b;
    HoanVi(p1, p2);
    cout << "p1: " << *p1 << endl;
    cout << "p2: " << *p2 << endl;
}
```

# TRUYỀN THAM SỐ CHO CON TRỎ

## 4. Nếu tham số truyền vào là 2 con trỏ thì sao?

```
void HoanVi(int *&p1, int *&p2)
{
    int *temp = p1;
    p1 = p2;
    p2 = temp;
}
int main()
{
    int a = 5, * p1 = &a;
    int b = 7, * p2 = &b;
    HoanVi(p1, p2);
    cout << "p1: " << *p1 << endl;
    cout << "p2: " << *p2 << endl;
}
```

# TRUYỀN THAM SỐ CHO CON TRỎ

## 5. Nếu tham số truyền vào là địa chỉ 2 con trỏ thì sao?

```
void HoanVi(int *&p1, int *&p2)
{
    int *temp = p1;
    p1 = p2;
    p2 = temp;
}
int main()
{
    int a = 5, * p1 = &a;
    int b = 7, * p2 = &b;
    HoanVi(&p1, &p2);
    cout << "p1: " << *p1 << endl;
    cout << "p2: " << *p2 << endl;
}
```

# CON TRỎ VÀ THAM SỐ HÀM

Hàm thay đổi này có bị mâu thuẫn với, mục 3 của Truyền Tham Số Cho Con Trỏ? Hãy đưa ra nhận xét của bạn.

```
void ThayDoi(int* p)
{
    *p = 10;
    cout << "1. Gia tri tai dia chi " << p << " la: " << *p <<
endl;
}

int main()
{
    int a = 5;
    ThayDoi(&a);
    cout << "2. Gia tri cua a la: " << a;
    return 0;
}
```

# CON TRỎ VÀ THAM SỐ HÀM

Quay trở lại với bài toán Hoán vị như vậy còn hàm này dùng để làm gì?

```
void HoanVi(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main()
{
    int a = 5;
    int b = 7;
    HoanVi(&a, &b);
    cout << a << endl;
    cout << b << endl;
    return 0;
}
```

# KIỂU TRẢ VỀ CON TRỎ

Hàm khởi tạo con trỏ mới có giá trị mới rồi trả về.

```
int* KhoiTao(int value)
{
    int *temp = new int;
    *temp = value;
    return temp;
}

int main()
{
    int* p = NULL;
    p = KhoiTao(10);
    cout << p << " " << *p << endl;
    cout << p << " " << *p << endl;
    return 0;
}
```

# KIỂU TRẢ VỀ CÒN TRỞ

Đơn giản hơn và kết quả tốt hơn?

```
int* KhoiTao(int value)
{
    int temp = value;
    return &temp;
}

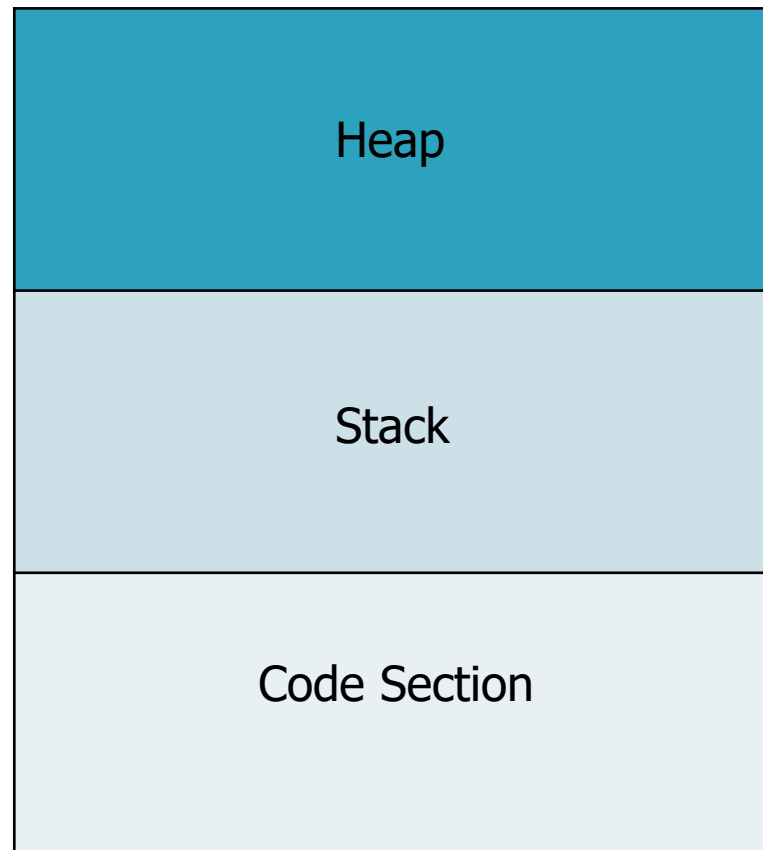
int main()
{
    int* p = NULL;
    p = KhoiTao(10);
    cout << p << " " << *p << endl;
    cout << p << " " << *p << endl;
    return 0;
}
```



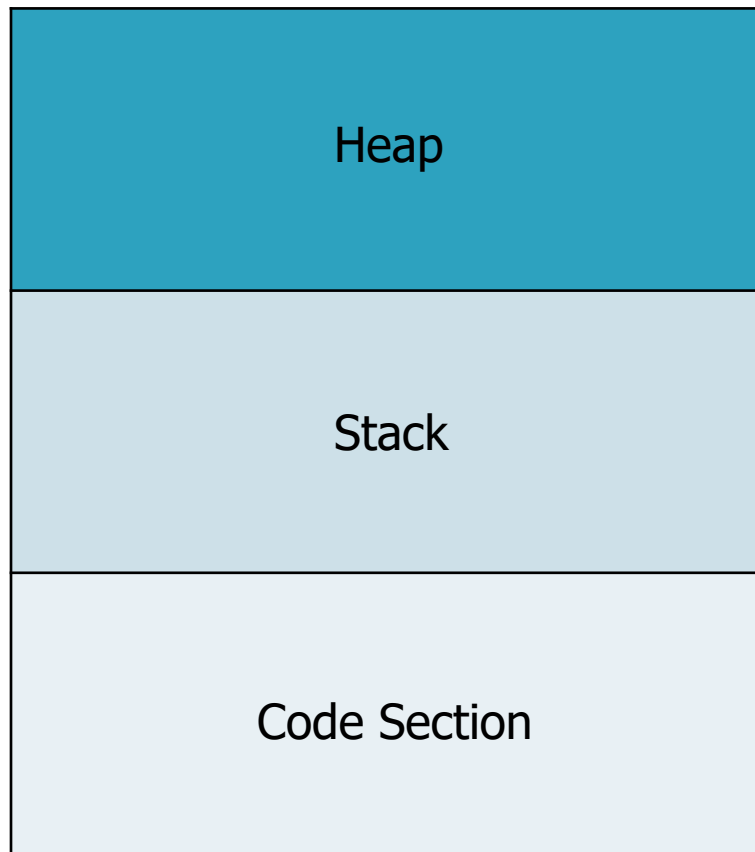
# VÙNG NHỚ STACK VÀ HEAP

```
int main()
{
    int a = 5;
    int* p = new int;
    delete p;
    return 0;
}
```

Khi nào sử dụng  
Stack, khi nào sử  
dụng Heap.



# VÙNG NHỚ STACK VÀ HEAP

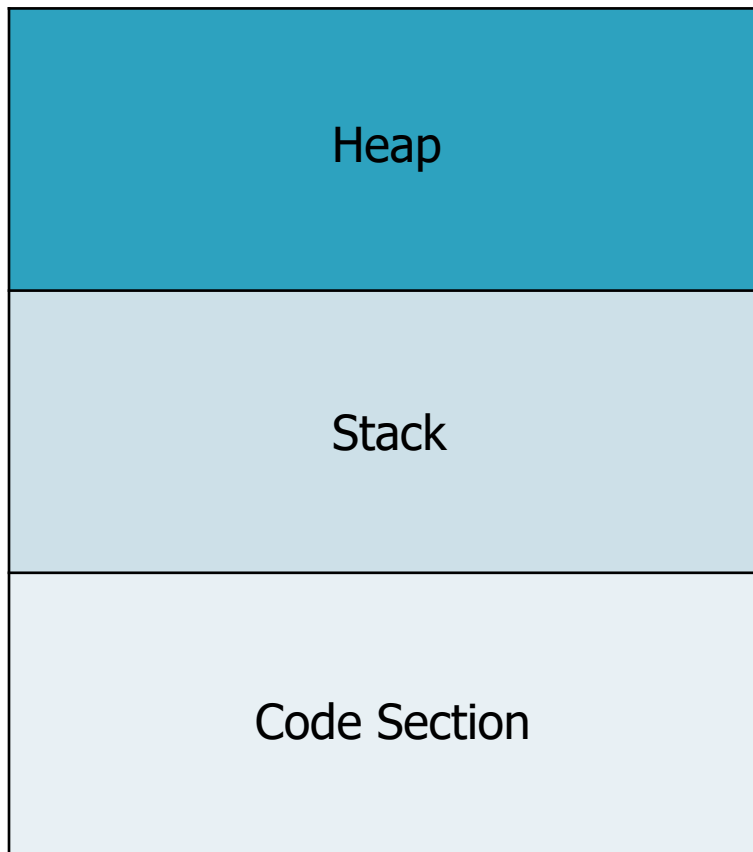


**1. Code Segment:** Nơi lưu trữ toàn bộ mã máy chương trình.

**2. Data Segment:** Nơi lưu trữ các biến chương trình với giá trị khác 0.

**3. BSS Segment:** Nơi lưu trữ các biến chương trình với giá trị chưa khởi tạo.

# VÙNG NHỚ STACK VÀ HEAP



## **Automatic Variables**

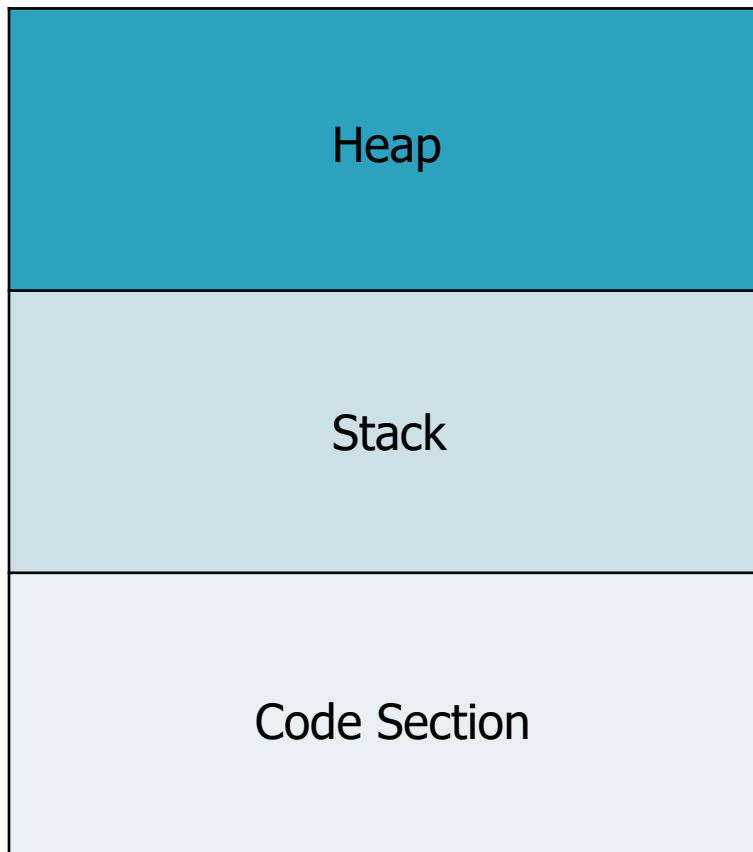
**Storage:** Vùng nhớ do CPU quản lý.

Thực hiện theo cơ chế LIFO.

Có dung lượng nhỏ giới hạn.

Dùng lưu trữ Function parameter, Local variables.

# VÙNG NHỚ STACK VÀ HEAP

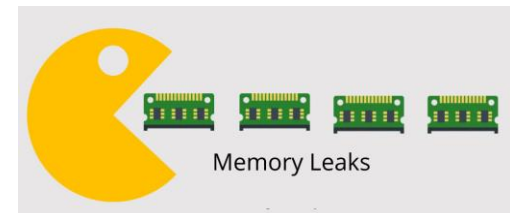


## Dynamic Memory

**Allocation:** Vùng nhớ do LTV quản lý.

C++: tạo vùng nhớ khi xài new, xóa khi xài delete.

Có dung lượng lớn hơn Stack.



# XÓA CON TRỎ SAU KHI SỬ DỤNG

Nếu không xóa con trỏ thì chuyện gì sẽ xảy ra?

```
#include <iostream>
using namespace std;
struct ToaDo
{
    float* x;
    float* y;
};
void KhoiTao()
{
    ToaDo a[100000];
    for (int i = 0; i < 100000; i++)
    {
        a[i].x = new float;
        a[i].y = new float;
        delete a[i].x;
        delete a[i].y;
    }
}
```

# XÓA CON TRỎ SAU KHI SỬ DỤNG

Nếu không xóa con trỏ thì chuyện gì sẽ xảy ra?

```
int main()
{
    for (int i = 0; i < 1000000; i++)
    {
        KhoiTao();
    }
    return 0;
}
```



Memory Leak

**SỬ DỤNG CON TRỎ VỚI CÁC CẤU  
TRÚC DỮ LIỆU KHÁC.**

# CON TRỎ MẢNG MỘT CHIỀU

Bạn có thể dùng con trỏ để sử dụng mảng 1 chiều động.

```
int main()
{
    int n = 5;
    int* a;
    NhapMang(a, n);
    XuatMang(a, n);
    return 0;
}
```



# CON TRỎ MẢNG MỘT CHIỀU

Bạn có thể dùng con trỏ để sử dụng mảng 1 chiều động.

```
void NhapMang(int*& a, int& n)
{
    cout << "Nhap so luong phan tu: ";
    cin >> n;
    a = new int[n];
    for (int i = 0; i < n; i++)
    {
        cout << "Nhap phan tu thu " << i << ": ";
        cin >> a[i];
    }
}
```

# CON TRỎ MẢNG MỘT CHIỀU

Bạn có thể dùng con trỏ để sử dụng mảng 1 chiều động.

```
void XuatMang(int* a, int n)
{
    cout<<"Nội dung của mảng là: ";
    for (int i = 0; i < n; i++)
        cout<< a[i] <<" ";
    cout << endl;
}
```

# CON TRỎ MẢNG MỘT CHIỀU

Hàm nhập mảng có thể được viết theo kiểu khác.

```
void NhapMang(int*& a, int& n)
{
    cout << "Nhap so luong phan tu: ";
    cin >> n;
    a = new int[n];
    for (int i = 0; i < n; i++)
    {
        cout << "Nhap phan tu thu " << i << ": ";
        cin >> *(a + i);
    }
}
```

# CON TRỎ MẢNG MỘT CHIỀU

Hàm nhập mảng có thể được viết theo kiểu khác.

```
void NhapMang(int*& a, int& n)
{
    cout << "Nhap so luong phan tu: ";
    cin >> n;
    a = new int[n];
    for (int i = 0; i < n; i++)
    {
        cout << "Nhap phan tu thu " << i << ": ";
        cin >> *(a++);
    }
}
```

# CON TRỎ MẢNG HÀM

Mỗi hàm đều có một địa chỉ, vì thế chúng ta có thể dùng con trỏ để trỏ đến một hàm và yêu cầu hàm đó xử lý.

```
int Ham()  
{  
    return 1;  
}  
int main()  
{  
    cout << Ham << endl;  
    return 0;  
}
```

# CON TRỎ MẢNG HÀM

Thực chất khi gọi một hàm chính ta là ta yêu cầu hệ điều hành hãy thực thi đoạn lệnh được lưu tại địa chỉ tương ứng.

```
int TinhTong(int a, int b)
{
    return a + b;
}
int main()
{
    int (* p)(int, int);
    p = TinhTong;
    int ketqua = p(5, 10);
    cout << ketqua;
    return 0;
}
```

# CON TRỎ MẢNG HAI CHIỀU

```
void Nhap(int**& a, int& m, int& n)
{
    cout << "Nhap m: ";
    cin >> m;
    cout << "Nhap n: ";
    cin >> n;
    a = new int* [m];
    for (int i = 0; i < m; i++)
        a[i] = new int[n];
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << "a[" << i << "][" << j << "]: ";
            cin >> a[i][j];
        }
    }
}
```

# CON TRỎ MẢNG HAI CHIỀU

```
void Xuat(int** a, int m, int n)
{
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << a[i][j] << " ";
        }
        cout << endl;
    }
}
```



# CON TRỎ MẢNG HAI CHIỀU

```
int main()
{
    int n, m;
    int** a;
    Nhap(a, m, n);
    Xuat(a, m, n);
    return 0;
}
```

# STRUCT CON TRỎ

Cho cấu trúc một sinh viên có Họ Tên, Điểm Toán, Điểm Văn. Hãy sử dụng cấu trúc sinh viên này như một con trỏ.

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

struct SinhVien
{
    string HoTen;
    float DiemToan;
    float DiemVan;
};
```

# STRUCT CON TRỎ

```
void Nhap(SinhVien* sv)
{
    cout << "Nhap ho ten: ";
    getline(cin, sv->HoTen);
    cout << "Nhap diem Toan: ";
    cin >> sv->DiemToan;
    cout << "Nhap diem Van: ";
    cin >> sv->DiemVan;
}
```

# STRUCT CON TRỎ

```
void Xuat(SinhVien* sv)
{
    cout << "Ho ten: " << sv->HoTen << endl;
    cout << "Diem Toan: " << sv->DiemToan << endl;
    cout << "Diem Van: " << sv->DiemVan << endl;
}
int main()
{
    SinhVien* sv = new SinhVien;
    Nhap(sv);
    Xuat(sv);
}
```

# STRUCT CON TRỎ

Cho cấu trúc một sinh viên có Họ Tên, Điểm Toán, Điểm Văn. Hãy sử dụng cấu trúc sinh viên này như một con trỏ. **(Sử dụng con trỏ bên trong cấu trúc)**

```
#include <iostream>
#include <string>
using namespace std;
struct SinhVien
{
    string *HoTen;
    float *DiemToan;
    float *DiemVan;
};
```

# STRUCT CON TRỎ

```
void Nhap(SinhVien* sv)
{
    cout << "Nhap ho ten: ";
    sv->HoTen = new string;
    getline(cin, *sv->HoTen);
    cout << "Nhap diem Toan: ";
    sv->DiemToan = new float;
    cin >> *sv->DiemToan;
    cout << "Nhap diem Van: ";
    sv->DiemVan = new float;
    cin >> *sv->DiemVan;
}
```

# STRUCT CON TRỎ

Cho cấu trúc một sinh viên có Họ Tên, Điểm Toán, Điểm Văn. Hãy sử dụng cấu trúc sinh viên này như một con trỏ. **(Sử dụng con trỏ bên trong cấu trúc, nhưng Cấu trúc thì không sử dụng con trỏ)**

```
#include <iostream>
#include <string>
using namespace std;
struct SinhVien
{
    string *HoTen;
    float *DiemToan;
    float *DiemVan;
};
```

# STRUCT CON TRỎ

```
void Nhap(SinhVien& sv)
{
    cout << "Nhap ho ten: ";
    sv.HoTen = new string;
    getline(cin, *sv.HoTen);
    cout << "Nhap diem Toan: ";
    sv.DiemToan = new float;
    cin >> *sv.DiemToan;
    cout << "Nhap diem Van: ";
    sv.DiemVan = new float;
    cin >> *sv.DiemVan;
}
```



## XÓA CON TRỎ SAU KHI SỬ DỤNG

Khi con trỏ là một biến kiểu dữ liệu thông thường sẽ xóa khác, khi con trỏ là một mảng động sẽ xóa khác.

```
int main()
{
    int *p = new int[5];
    *p = 5;
    cout << *p << endl;
    cout << &p << endl;
    p = NULL;
    delete p; //Xóa theo kiểu dữ liệu bình thường
    delete[] p; //Xóa khi p là mảng động kiểu con trỏ
}
```

# CON TRỎ CHUỖI KÝ TỰ

Cách khai báo và sử dụng.

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

int main()
{
    char* s = new char[20];
    strcpy(s, "ky thuat lap trinh");
    cout << s;
    return 0;
}
```

## CON TRỎ CHUỖI KÝ TỰ

- **strlen**: tính chiều dài của chuỗi.
- **strcpy**: chép một chuỗi vào chuỗi hiện tại.
- **strlwr/strupr**: biến chuỗi thành viết thường / viết hoa.
- **strrev**: đảo ngược ký tự trong chuỗi.
- **strcmp/stricmp**: so sánh chuỗi, phân biệt hoa thường / không phân biệt hoa thường.
- **strcat**: nối chuỗi.
- **strstr**: Tìm vị trí xuất hiện đầu tiên.