TRƯỜNG ĐẠI HỌC MỞ
THÀNH PHỐ HỒ CHÍ MINH

KỸ THUẬT LẬP TRÌNH

Chương 3: Con trỏ

1

Mục tiêu

- Sau khi học xong chương này, người học có thể:

1. Biết ý nghĩa và cách sử dụng con trỏ
2. Vận dụng được con trỏ vào hàm, mảng và cách cấp phát động

2

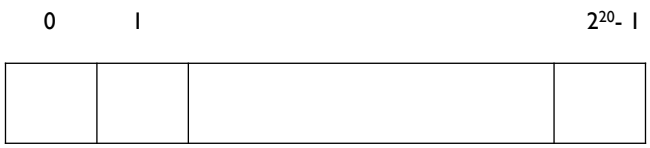
Nội dung

Con trỏ (Pointer)

1. Giới thiệu
2. Khai báo, khởi tạo con trỏ
3. Một số phép toán với con trỏ
4. Con trỏ và mảng
5. Biến cấp phát động
6. Con trỏ và hàm

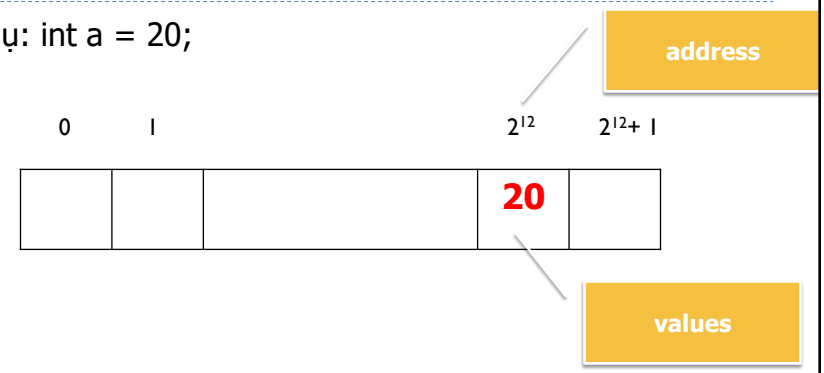
3

- Biến: là tên được định danh để thay thế cho vị trí trong bộ nhớ.
- Bộ nhớ máy tính là chuỗi các bytes (bắt đầu từ 0). Các con số này chính là địa chỉ (address).
- **Ví dụ**: 1 MB (megabyte)



4

- Ví dụ: int a = 20;



- Đối tượng lưu trữ địa chỉ (address) của biến được gọi là con trỏ (pointer)

- Cú pháp khai báo biến con trỏ:

```
dataType* variableName;
```

- Trong đó:
  - dataType: kiểu dữ liệu của biến mà con trỏ trỏ đến
  - variablename: tên biến con trỏ.

Ví dụ:

```
int* p;
```

- p: biến con trỏ tên p
- int: con trỏ p trỏ đến và lưu địa chỉ của dữ liệu kiểu số nguyên.

- Khởi tạo cho biến con trỏ:
  - Biến con trỏ phải được gán giá trị trước khi sử dụng;
  - Giá trị gán có thể chính là địa chỉ của 1 biến cụ thể hoặc là hằng NULL (0 hoặc nullptr);
  - Con trỏ được gán NULL tức là con trỏ null (null pointer).

Ví dụ: 3 khai báo và khởi tạo sau đây là tương đương nhau

```
int* ptr = NULL;
int* ptr = 0;
int* ptr = nullptr;
```


1. Giới thiệu
2. Khai báo và khởi tạo con trỏ
3. Một số phép toán với con trỏ
4. Con trỏ và mảng
5. Biến cấp phát động
6. Con trỏ và hàm

- Một số lưu ý khi khai báo biến con trỏ:

```
int* p1, p2;
```

p2 là 1 biến số nguyên

p1 là con trỏ lưu địa chỉ 1 dữ liệu kiểu số nguyên

- Nên khai báo mỗi biến con trỏ trên 1 dòng.
- Nên đặt tên biến con trỏ bắt đầu với tiền tố p hoặc ptr (pointer) để tiện kiểm soát.

- Toán tử & (address-of operator): lấy địa chỉ của 1 biến

Ví dụ 1:

```
int x = 20;
int* p;
p = &x;
cout << "Dia chi cua bien x la: " << p << endl;
```

C:\WINDOWS\system32\cmd.exe

```
Dia chi cua bien x la: 010FFA40
Press any key to continue . . .
```

Ví dụ 2:

```
double x = 20;
int* p;
p = &x;  //error
cout << "Dia chi cua bien x la: " << p << endl;
```


- Toán tử * (content-of operator/ dereferences operator): lấy nội dung của biến con trỏ trỏ đến.

Ví dụ:

```
int x = 20;
int* p;
p = &x;
cout << "Gia tri cua bien x la: " << *p << endl;
```

C:\WINDOWS\system32\cmd.exe

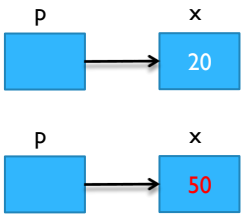
```
Gia tri cua bien x la: 20
Press any key to continue . . .
```

Ví dụ:

```
int x = 20;
int* p;
p = &x;

*p = 50;

cout << "Gia tri cua bien x la: " << x << endl;
```



C:\WINDOWS\system32\cmd.exe

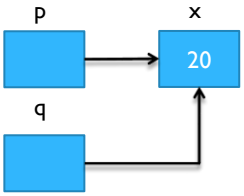
```
Gia tri cua bien x la: 50
Press any key to continue . . .
```


Phép gán

- Gán địa chỉ 1 biến khác cho biến con trỏ (cùng kiểu dữ liệu)
- Gán 2 biến con trỏ cùng kiểu dữ liệu với nhau.

Ví dụ:

```
int x = 20;
int* p, *q;
p = &x;
q = p;
cout << "Gia tri luu tru trong q la: " << *q <<
endl;
```

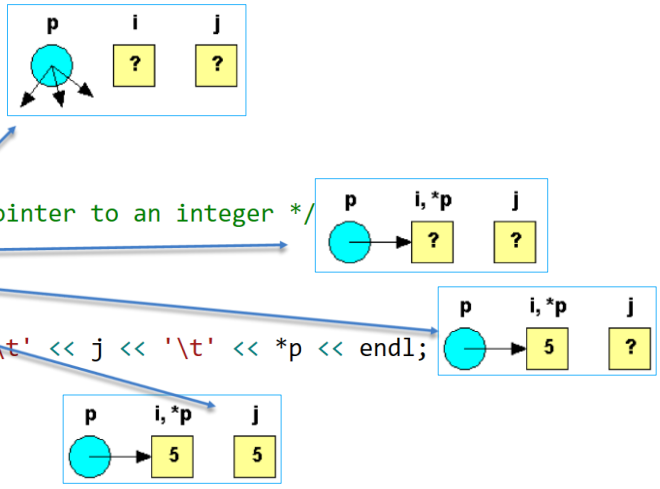


C:\WINDOWS\system32\cmd.exe

```
Gia tri luu tru trong q la: 20
Press any key to continue . . .
```

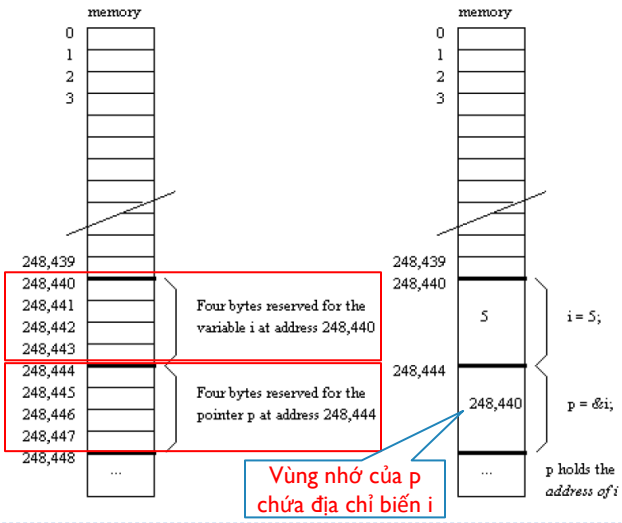
Ví dụ:

```
int main()
{
    int i, j;
    int *p; /* a pointer to an integer */
    p = &i;
    *p = 5;
    j = i;
    cout << i << '\t' << j << '\t' << *p << endl;
    return 0;
}
```



Ví dụ:

```
int main()
{
    int i, j;
    int *p; /* a p
    p = &i;
    *p = 5;
    j = i;
    cout << i << '
    return 0;
}
```



Phép so sánh:

2 biến con trỏ cùng kiểu dữ liệu có thể so sánh được với nhau.

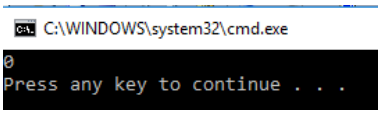
```
int* ptr1;
int* ptr2;
//........
```

Lệnh so sánh	Kết quả
ptr1 == ptr2	TRUE nếu 2 con trỏ cùng trỏ đến 1 vùng nhớ
ptr1 != ptr2	TRUE nếu 2 con trỏ không cùng trỏ đến 1 vùng nhớ
ptr1 != NULL	TRUE nếu con trỏ ptr1 khác NULL

Phép so sánh:

Ví dụ:

```
int* p, *q;
p = NULL;
int x =  2;
q =  &x;
bool kq = (p == q);
cout << kq << endl;
```



Phép so sánh:

Ví dụ:

```
int* p, *q;
p = NULL;
int x =  2;
q =  &x;
bool kq = (p != q);
cout << kq << endl;
```

C:\WINDOWS\system32\cmd.exe

```
1
Press any key to continue . . .
```


Phép so sánh:

Ví dụ:

```
int* p, *q;
int x =  2;
p = &x;
q =  &x;
bool kq = (p == q);
cout << kq << endl;
```

C:\WINDOWS\system32\cmd.exe

```
1
Press any key to continue . . .
```

Phép cộng, trừ

- Cộng (++), trừ (--) tương tự như biến bình thường, nhưng giá trị sẽ thay đổi theo kích thước kiểu dữ liệu của con trỏ.
- Có thể cộng, trừ giữa 2 biến con trỏ cùng kiểu dữ liệu, hay giữa con trỏ và 1 số nguyên.

1. Giới thiệu
2. Khai báo và khởi tạo con trỏ
3. Một số phép toán với con trỏ
4. Con trỏ và mảng
5. Biến cấp phát động
6. Con trỏ và hàm

Phép cộng, trừ

- **Ví dụ:**

```
int* p;
int x =  2;
p = &x;
cout << "Dia chi con tro p tro den: " << int(p) << endl;
p++;
cout << "Dia chi con tro p tro den sau khi tang: " << int(p) << endl;
```

C:\WINDOWS\system32\cmd.exe

```
Dia chi con tro p tro den: 16383680
Dia chi con tro p tro den sau khi tang: 16383684
Press any key to continue . . .
```

Phép cộng, trừ

- **Ví dụ:**

```
int* p;
int x =  2;
p = &x;
cout << "Dia chi con tro p tro den: " << int(p) << endl;
p += 2;
cout << "Dia chi con tro p tro den sau khi tang: " << int(p) << endl;
```

C:\WINDOWS\system32\cmd.exe

```
Dia chi con tro p tro den: 19921364
Dia chi con tro p tro den sau khi tang: 19921372
Press any key to continue . . .
```


1. Giới thiệu
2. Khai báo và khởi tạo con trỏ
3. Một số phép toán với con trỏ
4. Con trỏ và mảng
5. Biến cấp phát động
6. Con trỏ và hàm

LƯU Ý:

- Không được sử dụng biến con trỏ trỏ đến hằng.

Ví dụ:

```
const int x = 20;
int* p = &x;        //error
```

- Không nên lạm dụng con trỏ, sẽ làm câu lệnh phức tạp thêm

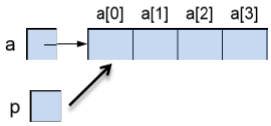
```
int x = 20;
int c = *(&x);
cout << c;
```

Con trỏ và mảng

- Tên mảng là con trỏ trỏ đến phần tử đầu tiên trong mảng

Ví dụ:

```
int a[4];
int *p;
p = a;
```



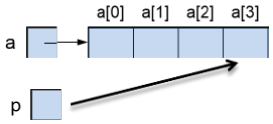
Tương đương với

```
p = a[0];
```


Con trỏ và mảng

Ký pháp độ dời:

```
p = p + 3;
```




```
int x = *(p + 3);
```
// x nhận giá trị của a[3], tương đương với *(a + 3)

Ký pháp chỉ số:

p[1] tương đương với a[1]

Con trỏ và mảng

```
int main()
{
    int a[] = {24 , 31, 19, 16};
    int n = sizeof(a) / sizeof(a[0]);
    int* p = a;

    //in mang dung ten mang va ky phap chi so
    cout << "Ten mang va ky phap chi so\n";
    for (int i = 0; i < n; i++)
        cout << "a[" << i << "] = " << a[i]
             << endl;
```


Con trỏ và mảng

```
    //in mang dung ten mang va ky phap do doi
    cout << "\nTen mang va ky phap do doi\n";
    for (int offset = 0; offset < n; offset++)
        cout << "*a( + " << offset << ") = "
             << *(a + offset) << endl;

    //in mang dung con tro va ky phap chi so
    cout << "\nCon tro va ky phap chi so\n";
    for (int i = 0; i < n; i++)
        cout << "p[" << i << "] = " << p[i]
             << endl;

    //in mang dung con tro va ky phap do doi
    cout << "\nCon tro va ky phap do doi\n";
    for (int offset = 0; offset < n; offset++)
        cout << "*(p + " << offset << ") = "
             << *(p + offset) << endl;
}
```

Con trỏ và mảng

Với khai báo:

```
int a[]= {24, 31, 19, 16};
int *p = a;
```

Các cách truy xuất giá trị phần tử sau đây là tương đương:

```
int x = a[2];
int x = *(a + 2);
int x = p[2];
int x = *(p + 2);
```

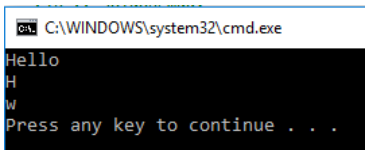

Mảng các con trỏ

- Là một mảng lưu trữ toàn các biến con trỏ.
- Thường dùng cho mảng chứa đối tượng chuỗi, hay mảng chứa đối tượng là ký tự.

Ví dụ:

```
char *a[2];
a[0] = "Hello";
a[1] = "world";

cout << a[0] << endl;
cout << *a[0] << endl;
cout << *a[1] << endl;
```



Bài tập

```
char str[] = "MyString";
char *pc = str;

cout << str[0] << ' ' << *pc << ' '
<< pc[3] << endl;
pc += 2;
cout << *pc << ' ' << pc[2] << ' '
<< pc[5]<<endl;
```


Bài tập

```
int a[] = { 1, 2, 3, 4, 5, 6, 7 };
int *pIn = a;
*pIn = 8;
*(pIn + 2) = 10;
for (int i = 0; i < 4 ; i++)
     cout << pIn[i] << '\t';
cout << endl;
pIn += 2;
for (int i = 0; i < 4; i++)
     cout << pIn[i] << '\t';
```

Biến cấp phát động

- Là biến được tạo trong khi chương trình đang thực thi.
- Cần biến cấp phát động ta dùng toán tử new
- Không cần sử dụng nữa dùng toán tử delete (hủy bỏ biến đã tạo và thu hồi vùng nhớ)

Toán tử new: cấp phát cho biến đơn hoặc cho mảng

Cú pháp cấp phát biến đơn:

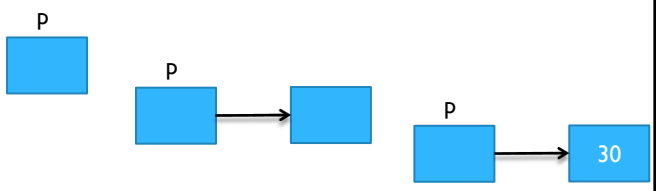
```
pointerName = new dataType;
```

Trong đó:

- pointername: tên con trỏ
- dataType: kiểu dữ liệu mà biến pointerName trỏ đến

Ví dụ:

```
int* p;
p = new int;
*p = 30;
```



Toán tử new: cấp phát cho biến đơn hoặc cho mảng

Cú pháp cấp phát mảng:

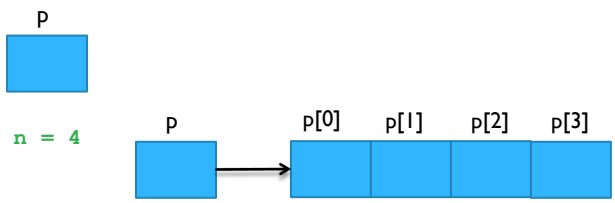
```
pointerName = new dataType[numberOfElement];
```

Trong đó:

- pointername: tên con trỏ
- dataType: kiểu dữ liệu mà biến pointerName trỏ đến
- NumberOfElement: số lượng phần tử của mảng cấp phát động.

Ví dụ:

```
int* p;
int n;
cin >> n; //gia su n = 4
p = new int[n];
```



1. Giới thiệu
2. Khai báo và khởi tạo con trỏ
3. Một số phép toán với con trỏ
4. Con trỏ và mảng
5. Biến cấp phát động
6. Con trỏ và hàm

Toán tử delete: thu hồi vùng nhớ đã cấp phát khi không còn sử dụng (*vùng nhớ của biến cấp phát động luôn phải được trả lại khi không còn sử dụng trong chương trình*).

Hủy bỏ cấp phát động cho biến:

```
delete pointerName;
```

Hủy bỏ cấp phát động cho mảng:

```
delete [] pointerName;
```

Lưu ý khi cấp phát động:

```
int* p;
p = new int;
*p = 50;
p = new int;
*p = 35;
```

//Không thể truy xuất đến biến đầu tiên lưu trữ giá trị 50

//nên hủy biến cấp phát động p ban đầu: delete p rồi hãy tiếp tục cấp phát động cho biến mới

Cấp phát động cho mảng 1 chiều

Cấp phát tĩnh	Cấp phát động
- Khai báo mảng với số lượng phần tử tối đa - Nhập số lượng phần tử thực tế làm việc - Nhập giá trị cho số lượng phần tử thực tế - Thực hiện tính toán…	- Nhập số lượng phần tử cần - Xin cấp phát đúng số lượng phần tử cần cho mảng - Nhập giá trị cho đúng số phần tử của mảng trong vùng nhớ - Thực hiện tính toán - Hủy vùng nhớ cấp phát - Đưa con trỏ về con trỏ rỗng

Mảng 1 chiều cấp phát động

```
int n;
cout << "Nhap so luong phan tu can: ";
cin >> n;
int* a;
a =  new int[n];
//Nhap mang
for  (int i = 0; i < n; i++)
{
  cout << "Nhap gia tri cho phan tu thu " << i + 1
  << ": ";
  cin >> a[i];
}
```


Mảng 1 chiều cấp phát động

```
//Xuat mang
cout << "\nMang luu tru la: ";
for (int i = 0; i < n; i++)
      cout << a[i] << "\t";

cout << endl;
//Huy cap phat dong, dua con tro ve con tro rong
delete []a;
a = nullptr;
```

Cấp phát động cho mảng 2 chiều

Cấp phát tĩnh	Cấp phát động

- Khai báo mảng với số lượng phần tử tối đa cho mỗi chiều
- Nhập số lượng phần tử thực tế làm việc ở mỗi chiều
- Nhập giá trị cho số lượng phần tử thực tế tương ứng
- Thực hiện tính toán
	- Nhập số lượng phần tử cần ở mỗi chiều
	- Xin cấp phát đúng số lượng phần tử cần cho mảng:
	  - Cấp phát 1 mảng các con trỏ (số dòng)
	  - Cấp phát vùng nhớ cho mỗi con trỏ trên từng dòng (số cột)
	- Nhập giá trị cho từng phần tử của mảng trong vùng nhớ
	- Thực hiện tính toán
	- Hủy vùng nhớ cấp phát (theo vùng nhớ cho con trỏ rồi đến hủy vùng nhớ cho mảng)
	- Đưa con trỏ về con trỏ rỗng

Mảng 2 chiều cấp phát động

```
int r, c;
cout << "Nhap so dong va so cot can: "; cin >> r >> c;
int **a;
//Cap phat mang cac con tro (so dong)
a = new int* [r];
//Cap phat vung nho cho moi con tro (so cot)
for (int i = 0; i < r; i++)
      a[i] = new int [c];
```

Mảng 2 chiều cấp phát động

```
//Nhap mang
for (int i = 0; i < r; i++)
{
  cout << "\nNhap " << c << " gia tri cho dong thu " << i + 1 <<
  ": ";
  for (int j = 0; j < c; j++)
       cin >> a[i][j];
}
//Xuat mang
cout << "\nMang dang luu tru: " << endl;
for (int i = 0; i < r; i++)
{
  for (int j = 0; j < c; j++)
       cout << a[i][j] << "\t";
  cout << endl;
}
```


1. Giới thiệu
2. Khai báo và khởi tạo con trỏ
3. Một số phép toán với con trỏ
4. Con trỏ và mảng
5. Biến cấp phát động
6. Con trỏ và hàm

Mảng 2 chiều cấp phát động

```
//huy vung nho da cap phat

for (int i = 0; i < r; i++)

      delete [] a[i];


delete [] a;

a = nullptr;
```

Bài tập

1. Viết chương trình dùng cấp phát động cho nhập vào mảng 1 chiều gồm các số nguyên (tối đa 20 phần tử). Sau đó tiến hành đảo ngược mảng vừa nhập. Xuất lại mảng cho người dùng kiểm tra.

2. Viết chương trình dùng cấp phát động để nhập, xuất 1 mảng số nguyên gồm m hàng và n cột. Đếm xem mảng đang lưu trữ có bao nhiêu số là số nguyên tố?

1. Giới thiệu
2. Khai báo và khởi tạo con trỏ
3. Một số phép toán với con trỏ
4. Con trỏ và mảng
5. Biến cấp phát động
6. Con trỏ và hàm

Qui trình chuyển cấp phát động cho mảng 2 chiều bằng hàm

Function prototype:

```
void nhap (int **a, int r, int c);
void xuat (int **a, int r, int c);
```

Hàm main:

```
//Nhap r, c
//Cap phat mang con tro (so dong)
//Cap phat vung nho cho moi con tro (so cot)
//Goi ham nhap
```

Tham số hàm có kiểu con trỏ

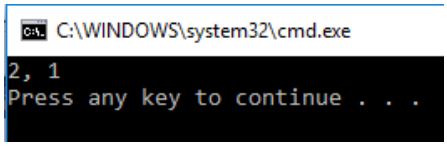
Được truyền theo 2 hình thức: tham trị và tham chiếu

- Tham trị: truyền địa chỉ.
- Tham chiếu: dùng dấu & sau dấu * của con trỏ. Thường được ưu tiên khi truyền có cấp phát động.

1. Giới thiệu
2. Khai báo và khởi tạo con trỏ
3. Một số phép toán với con trỏ
4. Con trỏ và mảng
5. Biến cấp phát động
6. Con trỏ và hàm

Tham số hàm có kiểu con trỏ truyền theo tham trị

```
void hoandoi(int* a, int* b)
{
  int tam = *a;
  *a = *b;
  *b = tam;
}
int main()
{
  int x = 1;
  int y = 2;
  int* p = &x;
  int* q = &y;
  hoandoi(p, q);
  cout << x << ", " << y << endl;
  return 0;
}
```



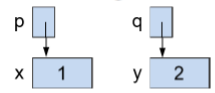
Tham số hàm có kiểu con trỏ truyền theo tham chiếu

```
void hoandoi(int*& a, int*& b)
{
  int tam = *a;
  *a = *b;
  *b = tam;
}
int main()
{
  int x = 1;
  int y = 2;
  int* p = &x;
  int* q = &y;
  hoandoi(p, q);
  cout << x << ", " << y << endl;
  return 0;
}
```

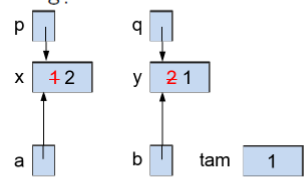
C:\WINDOWS\system32\cmd.exe

```
2, 1
Press any key to continue . . .
```

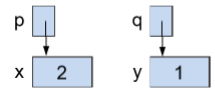

- Trước khi gọi hoandoi:



- Khi gọi hoandoi:



- Sau khi hoandoi thực hiện xong:



Tham số hàm có kiểu con trỏ truyền KHÔNG theo tham chiếu

```
void nhap(int* a, int n)
{
  a = new int[n];
  for (int i = 0; i < n; i++)
  {
    cout << "a[" << i << "]= ";
    cin >> a[i];
  }
}
void xuat(const int* a, int n)
{
  cout << "\nCac phan tu trong mang la: ";
  for (int i = 0; i < n; i++)
  {
      cout << a[i] << " ";
  }
  cout << endl << endl;
}
```


Tham số hàm có kiểu con trỏ truyền KHÔNG theo tham chiếu

```
int main()
{
  int* p;
  int n;
  cout << "Nhap so phan tu: ";
  cin >> n;

  p = new int[n];
  nhap(p, n);
  xuat(p, n);
}
```

C:\WINDOWS\system32\cmd.exe

```
Nhap so phan tu: 3
a[0]= 1
a[1]= 3
a[2]= 1

Cac phan tu trong mang la: -842150451 -842150451 -842150451

Press any key to continue . . .
```

Tham số hàm có kiểu con trỏ truyền KHÔNG theo tham chiếu

Nếu cấp phát động không ở trong hàm nhập khi truyền mảng (con trỏ) theo kiểu tham trị thì mảng xuất bình thường (nhờ vào giá trị địa chỉ)

```
void nhap(int* a, int n)
{
        //a = new int[n];
        for (int i = 0; i < n; i++)
        {
                cout << "a[" << i << "]= ";
                cin >> a[i];
        }
}
```

```
C:\WINDOWS\system32\cmd.exe
Nhap so phan tu: 2
a[0]= 6
a[1]= 7

Cac phan tu trong mang la: 6 7

Press any key to continue . . .
```


1. Giới thiệu
2. Khai báo và khởi tạo con trỏ
3. Một số phép toán với con trỏ
4. Con trỏ và mảng
5. Biến cấp phát động
6. Con trỏ và hàm

Tham số hàm có kiểu con trỏ truyền theo tham chiếu

```
void nhap (int*& a, int n);

void xuat (int* a, int n);
```

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

Tham số hàm có kiểu con trỏ truyền theo tham chiếu

```
void nhap (int* &a, int n);
void xuat (int* a, int n);
int main()
{
  int* p;
  int n;
  cout << "Nhap so phan tu: ";
  cin >> n;
  p = new int[n];
  nhap(p, n);
  xuat(p, n);
}
```

C:\WINDOWS\system32\cmd.exe

```
Nhap so phan tu: 3
a[0]= 1
a[1]= 2
a[2]= 3

Cac phan tu trong mang la: 1 2 3

Press any key to continue . . .
```


- Phân biệt tham số hàm có kiểu con trỏ truyền theo tham chiếu và không theo tham chiếu

Không truyền bằng tham chiếu	Truyền bằng tham chiếu
Sự thay đổi về tham chiếu không ảnh hưởng đến đối số truyền vào	Sự thay đổi về tham chiếu ảnh hưởng đến đối số truyền vào

void nhap(int *pArr, int n)
{
for (int i = 0; i < n; i++) {
cout << "Phan tu thu " << i << ": ";
cin >> pArr[i];
}
pArr = new int[2]{ 4, 5 };
}
	void nhap(int *&pArr, int n)
	{
	for (int i = 0; i < n; i++) {
	cout << "Phan tu thu " << i << ": ";
	cin >> pArr[i];
	}
	pArr = new int[2] { 4, 5 };
	}

Nhập n=3, các phần tử là 1, 2, 3 pArr = { 1, 2, 3 };	Nhập n=3, các phần tử là 1, 2, 3 pArr = { 4, 5 };

Hàm trả về con trỏ

Hàm trả về giá trị nhỏ nhất của 1 mảng số nguyên bằng cách dùng con trỏ.

```
//Ham tim phan tu nho nhat trong mang
//Nhan vao: mang so nguyen a, so phan tu n
//Tra ve: con tro den phan tu nho nhat
double* nhonhat(double a[], int n)
{
    int vitri_nhonhat = 0;
    for (int i = 1; i < n; i++)
        if (a[i] < a[vitri_nhonhat])
            vitri_nhonhat = i;
    return &a[vitri_nhonhat];
}
```


1. Giới thiệu
2. Khai báo và khởi tạo con trỏ
3. Một số phép toán với con trỏ
4. Con trỏ và mảng
5. Biến cấp phát động
6. Con trỏ và hàm

Hàm trả về con trỏ

Hàm trả về giá trị nhỏ nhất của 1 mảng số nguyên bằng cách dùng con trỏ.

```
int main()
{
    double arr[] = {11.0, 23.0, 13.0, 4.0,
                    57.0, 36.0, 317.0, 88.0,
                    9.0, 100.0, 121.0, 12.0};
    int arrsize = sizeof(arr)/sizeof(arr[0]);

    double* p = nhonhat(arr,arrsize);

    cout << "Phan tu nho nhat la " << *p << endl;

    return 0;
}
```

1. Giới thiệu
2. Khai báo và khởi tạo con trỏ
3. Một số phép toán với con trỏ
4. Con trỏ và mảng
5. Biến cấp phát động
6. Con trỏ và hàm

Bài tập

4. Viết chương trình dùng cấp phát động xây dựng hàm nhập và xuất 1 mảng số nguyên tối đa 15 phần tử. Sau đó xây dựng hàm kiểm tra xem mảng vừa nhập có phải là mảng đối xứng hay không? Viết chương trình kiểm tra các hàm trên.

5. Viết chương trình dùng cấp phát động để xây dựng hàm nhập, xuất 1 mảng số nguyên gồm m hàng và n cột, hàm trả về vị trí lưu trữ của giá trị đầu tiên trong mảng là số nguyên tố. Viết chương trình kiểm tra các hàm trên.

Q & A