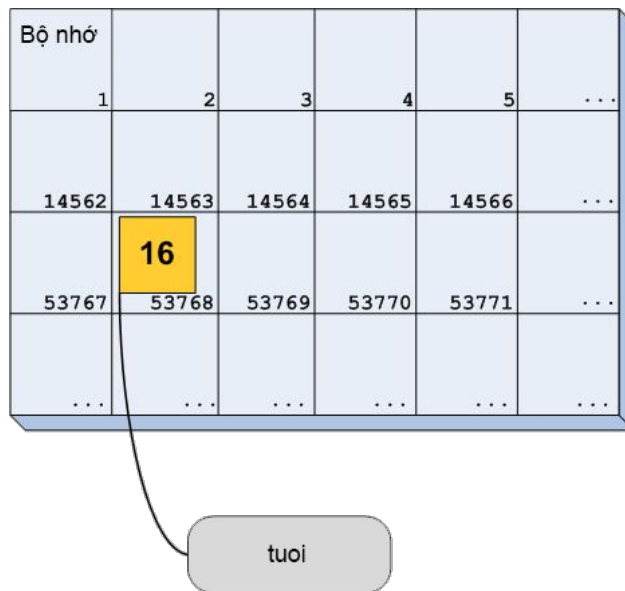


# Con trỏ trong C++

Nguyễn Đức Thắng

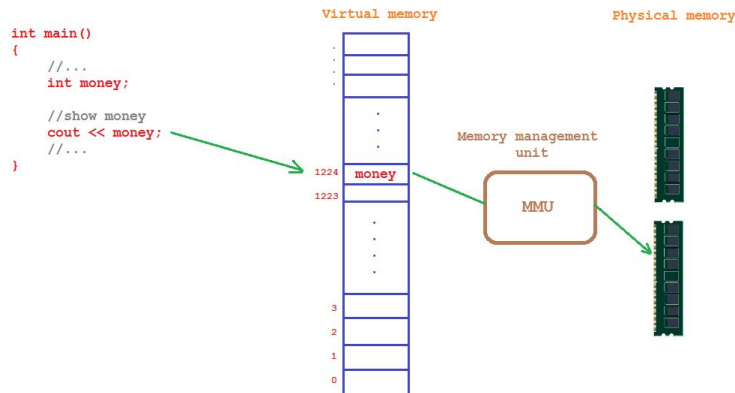
# Địa chỉ của biến

- Khi thao tác với **biến (variable)**, chúng ta *không cần quan tâm đến địa chỉ* của biến mà *chỉ cần quan tâm đến định danh* của biến.



# Virtual memory and Physical memory

- Việc truy xuất vào bộ nhớ thông thường phải qua các bước trung gian, người dùng thường không được truy xuất vào các ô nhớ trên thiết bị lưu trữ.
- Chúng ta chỉ có thể truy xuất đến vùng nhớ ảo, việc truy xuất bộ nhớ vật lý được thực hiện bởi thiết bị phần cứng **Memory management unit (MMU)**, và một chương trình định vị gọi là **Virtual address space**



# Variable address and address-of operator

- Địa chỉ của biến mà chúng ta nhìn thấy thực ra chỉ là những giá trị trên Virtual memory (t toán tử **&**).

```
int x = 5;  
std::cout << x << '\n'; // print the value of variable x  
std::cout << &x << '\n'; // print the memory address of variable x
```

- **Tham chiếu (reference)**: Tạo ra 1 biến khác có cùng kiểu dữ liệu, dùng chung vùng nhớ. Biến tham chiếu có địa chỉ cố định sau khi khởi tạo, không khởi tạo lại được.

```
int i1 = 10;  
int &i_ref = i1;           //reference to i1, not means address of i1  
  
cout << &i1 << endl;      //get address of i1  
cout << &i_ref << endl;   //get address of i_ref
```

# Variable address and address-of operator

- Toán tử trở đến (**dereference operator**) hay còn gọi là **indirection operator** (toán tử điều hành gián tiếp) được ký hiệu bằng dấu \* cho phép lấy giá trị vùng nhớ.

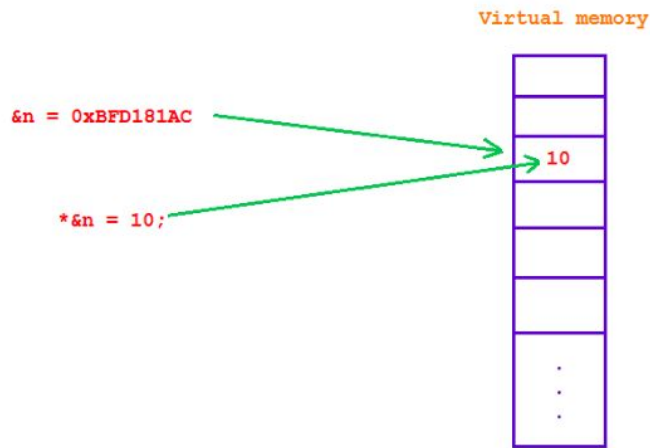
```
int n = 5;  
  
cout << n << endl;    //print the value of variable n  
cout << &n << endl;   //print the virtual memory address of variable n  
cout << *(&n) << endl; //print the value at the virtual memory address of variable n
```

- Còn có thể thay đổi giá trị bên trong vùng nhớ.

```
int n = 5;  
cout << n << endl;  
*(&n) = 10;  
cout << n << endl;
```

# Variable address and address-of operator

- Khác với tham chiếu, toán tử trở đến không tạo ra một tên biến khác, mà nó truy xuất trực tiếp đến vùng nhớ có địa chỉ cụ thể trên Virtual memory.

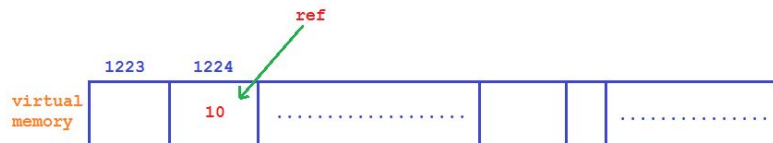


# Con trỏ (Pointer)

- Một con trỏ (pointer) là một biến dùng để lưu trữ địa chỉ của biến khác.
- Khác với tham chiếu, con trỏ có địa chỉ độc lập so với biến mà nó trỏ đến, giá trị bên trong vùng nhớ của con trỏ chính là địa chỉ của biến mà nó trỏ tới.

Reference example

```
int n = 10; //suppose address of n is 1224  
int &ref = n;
```



Pointer example




# Khai báo con trỏ

- Tổng quát: `<data_type> *<name_of_pointer>;`

- Ví dụ:

```
int *iPtr;  
float *fPtr;  
double *dPtr;  
  
int *iPtr1, *iPtr2;
```

- **Chú ý:** Dấu \* trong khai báo con trỏ không phải là toán tử trỏ đến, chỉ là cú pháp do C/C++ quy định.





# Khai báo con trỏ

- Cách khai báo dễ gây nhầm lẫn:

```
int *iPtr1; //We recommended you use this way to declare pointers
```

```
int* iPtr2, iPtr3;
```

- Kiểu dữ liệu của con trỏ không mô tả giá trị địa chỉ được lưu trữ bên trong con trỏ, mà kiểu dữ liệu của con trỏ xác định kiểu dữ liệu của biến mà nó trỏ tới.

```
cout << sizeof(char*) << endl;  
cout << sizeof(int*) << endl;  
cout << sizeof(double*) << endl;  
cout << sizeof(string*) << endl;
```

# Gán giá trị cho con trỏ

- Giá trị mà con trỏ lưu trữ là địa chỉ của biến khác có cùng kiểu dữ liệu.

```
int main()
{
    int value = 5;
    int *ptr = &value;

    cout << &value << endl;
    cout << ptr << endl;

    system("pause");
    return 0;
}
```



# Gán giá trị cho con trỏ

- Chúng ta có thể gán 2 con trỏ cùng kiểu.

```
int main()
{
    int value = 5;
    int *ptr1, *ptr2;

    ptr1 = &value; //ptr1 point to value
    ptr2 = ptr1;    //assign value of ptr1 to ptr2

    cout << ptr1 << endl;
    cout << ptr2 << endl;

    system("pause");
    return 0;
}
```

# Gán giá trị cho con trỏ

- Khác với tham chiếu, một con trỏ có thể trỏ đến địa chỉ khác trong bộ nhớ ảo sau khi được gán giá trị.

```
int main()
{
    int *ptr;

    int arr[5] = { 1, 2, 3, 4, 5 };

    for(int i = 0; i < 5; i++)
    {
        ptr = &arr[i];
        cout << ptr << endl;
    }

    system("pause");
    return 0;
}
```

# Các phép gán không hợp lệ

```
int iValue = 0;  
float fValue = 0.0;  
  
int *i_ptr = fValue;    //wrong! int pointer cannot point to the address of a double variable  
float *f_ptr = iValue;  //wrong! float pointer cannot point to the address of an int variable
```

```
int *ptr = 1245052; //wrong!
```

```
int *ptr = 0012FF7C; //wrong!
```

**Kết luận:** Chỉ có giá trị kiểu con trỏ (thông qua toán tử address-of) hoặc từ 1 biến con trỏ khác mới có thể được gán cho biến con trỏ.

# Truy xuất giá trị mà con trỏ trỏ đến

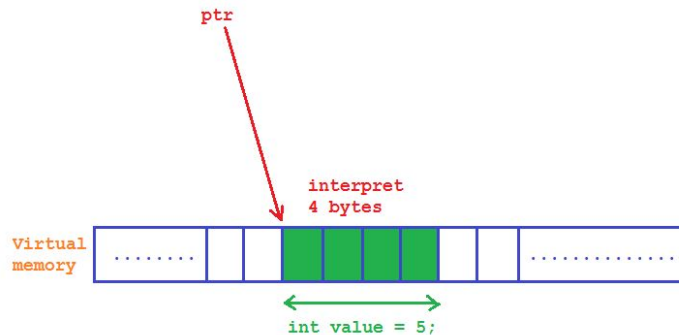
- Khi có một con trỏ trỏ đến một vùng nhớ, chúng ta có thể truy xuất giá trị tại địa chỉ đó bằng **toán tử trỏ đến (dereference operator)**
- 1 con trỏ có thể truy xuất nhiều vùng nhớ và sửa giá trị các vùng.

```
int *ptr; //declare an int pointer
int value = 5;

ptr = &value; //ptr point to value

cout << &value << endl; //print the address of value
cout << ptr << endl;    //print the address of value which is held

cout << value << endl;  //print the content of value
cout << *(&value) << endl; //print the content of value
cout << *ptr << endl;    //print the content of value
```



# NULL

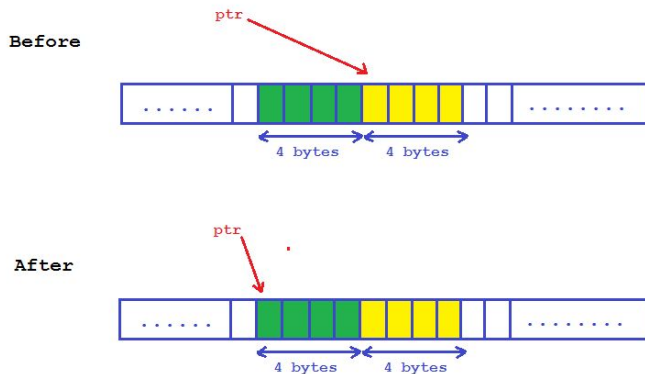
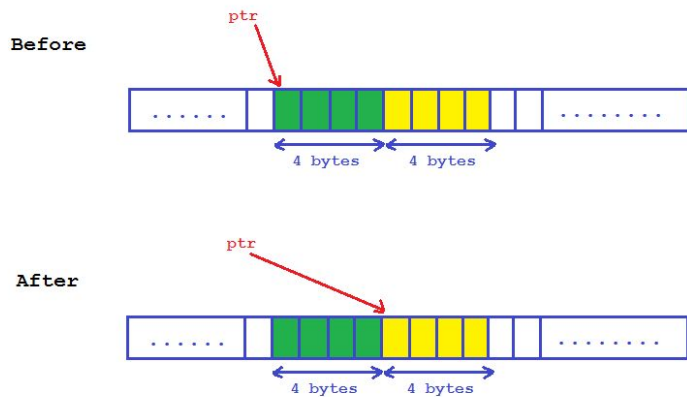
- Với con trỏ, **NULL** là một giá trị đặc biệt = con trỏ chưa được gán giá trị.
- Chuẩn C++11 cung cấp từ khóa **nullptr** tương tự NULL

```
int *ptr = NULL;

if(ptr == NULL)
{
    cout << "Do nothing" << endl;
}
else
{
    cout << *ptr << endl;
}
```

# Toán tử tăng giảm dùng trong con trỏ

- Toán tử **++** làm con trỏ trở đến vùng nhớ tiếp theo. Tương tự toán tử **--**





# Toán tử tăng giảm dùng trong con trỏ

- Toán tử **+** cho phép chúng ta trỏ đến vùng nhớ bất kì phía sau địa chỉ mà con trỏ đang nắm giữ. Toán tử **+** chỉ cho phép thực hiện với số nguyên.
- Tương tự với toán tử **-**

```
int value = 0;  
int *ptr = &value;  
  
cout << ptr << endl;  
  
ptr = ptr + 5;  
cout << ptr << endl;
```

- Chúng ta cũng có thể thực hiện các phép toán so sánh với con trỏ.

# Một số lưu ý khi sử dụng toán tử trong con trỏ

- Cần có cách sử dụng hợp lý tránh gây nhầm lẫn.
- Lệnh **\*p++**: truy xuất đến vùng nhớ mà p trỏ đến, sau đó trỏ đến địa chỉ tiếp theo.
- Lệnh **\*p\*n**: Nhân giá trị \*p với n
- Để rõ ràng, khi thao tác với con trỏ nên **thêm dấu ngoặc ( )**

```
int n = 5;  
int *p = &n; //p point to n  
  
*p++;  
p++;  
  
int n2 = *p*n;
```



# Con trỏ và mảng

- Địa chỉ mảng 1 chiều là địa chỉ phần tử đầu tiên của mảng. Vì thế có thể dùng con trỏ để truy xuất mảng 1 chiều.
- Con trỏ có thể trỏ đến 1 phần tử của mảng 1 chiều.
- Con trỏ cũng dùng toán tử [ ] để truy xuất được.

```
int arr[] = { 32, 13, 66, 11, 22 };
```

```
cout << *(arr) << endl;  
cout << *(arr + 1) << endl;  
cout << *(arr + 2) << endl;  
cout << *(arr + 3) << endl;  
cout << *(arr + 4) << endl;
```

```
int *ptr = arr; //ptr point to &arr[0]  
for (int i = 0; i < 5; i++)  
{  
    cout << *(ptr + i) << " ";  
}
```

```
int *ptr = arr;  
for (int i = 0; i < 5; i++)  
{  
    cout << ptr[i] << " ";  
}
```

# Sự khác nhau giữa con trỏ và mảng 1 chiều

- Sau khi con trỏ đến mảng 1 chiều, chúng ta có thể sử dụng tên con trỏ thay cho mảng 1 chiều. Tuy vậy, chúng vẫn khác nhau nếu chúng ta dùng toán tử **sizeof**
- Khi sử dụng mảng 1 chiều, toán tử sizeof trả về kích thước của mảng, từ đó có thể biết được số lượng phần tử của mảng. Còn con trỏ không làm được điều này.
- Ngoài ra, mảng 1 chiều khai báo cố định 1 vùng nhớ cố định trên bộ nhớ ảo, con trỏ sau khi trỏ vào mảng 1 chiều vẫn có thể trỏ đi vùng nhớ khác.



# Con trỏ và mảng kí tự

- Chuỗi kí tự có thể coi như mảng 1 chiều.

- Ví dụ:

```
char my_name[] = "Nguyen Duc Thang";
```

```
char *p_name = my_name;
```

```
p_name[1] = 'E';
```

```
cout << my_name << endl;
```

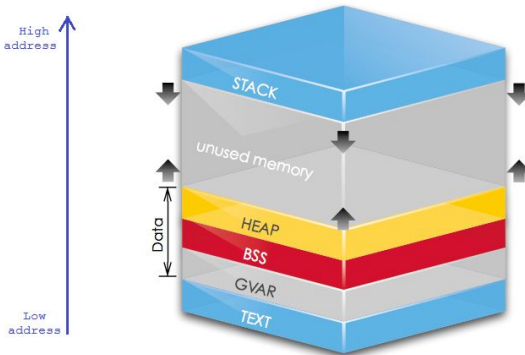
- **Chú ý:** Không thể in ra địa chỉ biến kí tự:

```
char ch = 'A';  
cout << &ch << endl;
```

# Cấp phát bộ nhớ động

- Có 2 cách thức cấp phát bộ nhớ trên bộ nhớ ảo:

1. Cấp phát bộ nhớ tĩnh: áp dụng cho biến static và biến toàn cục. (sử dụng phân vùng stack)
2. Cấp phát bộ nhớ động: Cấp phát biến cục bộ, tham số hàm. (Sử dụng phần vùng heap)



# Cấp phát bộ nhớ động cho biến

- Vùng nhớ cấp phát trên Heap không tự động hủy khi chương trình kết thúc, do lập trình viên quản lý.
- Toán tử cấp phát: **new**

```
int *p_int = new int;  
cout << "Put value into memory area" << endl;  
cin >> *p_int;  
cout << "Value at " << p_int << " is " << *p_int << endl;
```

- Có thể vừa cấp phát vừa khởi tạo giá trị tại vùng nhớ cho 1 biến đơn.

```
int *p1 = new int(5);  
int *p2 = new int { *p1 };
```

# Giải phóng vùng nhớ cấp phát cho biến

- Để giải phóng vùng nhớ được cấp phát, dùng toán tử: **delete**

```
int *p = new int;  
  
//using memory area at p  
//and then set it free  
  
delete p;
```

- Trong 1 số trường hợp, cấp phát heap thất bại, nếu không muốn ngoại lệ, ta dùng:

```
int *p = new (std::nothrow) int;
```

khi đó con trỏ nhận giá trị null nếu cấp phát thất bại.



# Cấp phát bộ nhớ động cho mảng

- Cũng sử dụng toán tử new, trả về địa chỉ phần tử đầu tiên nếu cấp phát thành công.

```
int *p_arr = new int[10];

//using this memory area
for (int i = 0; i < 10; i++)
{
    //Set value for each element
    cin >> *p_arr[i];
}
```

```
int num_of_elements;
cout << "Enter number of elements you want to create: ";
cin >> num_of_elements;

if(num_of_elements > 0)
    int *p_arr = new int[num_of_elements];
```

- Giải phóng:

```
int *p_arr = new int[10];

//.....

delete[] p_arr;
```



# Thay đổi kích thước vùng nhớ được cấp phát

- Cấp phát lại vùng nhớ mới
- (Copy dữ liệu vùng nhớ cũ sang vùng nhớ mới nếu cần)
- Giải phóng vùng nhớ cũ
- Cho con trỏ trỏ đến vùng nhớ mới



# Con trỏ và hằng

- Con trỏ cũng là 1 biến thông thường mà giá trị của nó là địa chỉ của biến khác. Vì thế từ khóa **const** cũng có thể sử dụng cho con trỏ như các kiểu dữ liệu khác.

```
const int value = 5;  
int *ptr = &value; //compile error
```

```
const int value = 5;  
const int *ptr = &value; //it's ok, ptr point to a "const int"  
*ptr = 10; //compile error
```

- Gán con trỏ cho 1 biến hằng, mặc dù chưa thực hiện thay đổi gì những compiler cũng ngăn chặn phép gán này để đảm bảo an toàn.

- Con trỏ hằng có thể trỏ vào biến hằng, nhưng không thể thay đổi giá trị bên trong vùng nhớ hằng.

- Con trỏ hằng cũng có thể trỏ đến vùng nhớ không phải là hằng số, nhưng không được thay đổi giá trị vùng nhớ đó.

```
int value = 5;  
const int *ptr = &value;  
*ptr = 10; //compile error
```

# Con trỏ và hằng

- **Pointer to const** là một con trỏ **trỏ đến biến hằng** chứ bản chất không phải là hằng số. Vì thế, sau khi khởi tạo nó vẫn có thể trỏ đến vùng nhớ khác.

```
const int *ptr = NULL;
```

```
int value1 = 5;  
ptr = &value1;
```

```
int value2 = 10;  
ptr = &value2;
```



const memory



non-const memory

```
const int *p;  
const int value = 10;  
p = &value;
```



# Con trỏ và hằng

- Làm thế nào để tạo con trỏ chỉ được gán giá trị 1 lần khi khởi tạo?

=> **Const pointer**: Đặt từ khóa const giữa dấu \* và tên con trỏ.

```
int value1 = 5;  
int value2 = 10;  
int *const ptr = &value1;  
ptr = &value2; //compile error
```

- Tuy nhiên, nếu vùng nhớ không là hằng, thì const pointer vẫn có thể thay đổi giá trị vùng nhớ này.

```
int value = 5;  
int *const ptr = &value;  
*ptr = 10; //it's ok, change value of non-const memory area
```

# Con trỏ và hằng

- Chúng ta có thể kết hợp **pointer to const** và **const pointer** để tạo thành **const pointer to const**. Loại con trỏ này có chức năng read only và nó cũng không trỏ được đến vùng nhớ khác sau khi khởi tạo.

```
int value = 5;  
const int *const ptr = &value;  
  
&ptr = 10; //compile error  
  
int otherValue = 10;  
ptr = &otherValue; //compile error
```

# Bài tập (Đoạn code nào biên dịch được?)

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

int main() {

    char str[] = "Nguyen Duc Thang";
    char *const p_str = str;

    for(int i = 0; i < strlen(str); p_str++)
    {
        *p_str = ' ';
    }

    cout << p_str << endl;

    return 0;
}
```

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

int main() {

    char str[] = "Nguyen Duc Thang";
    const char *p_str = str;

    for(int i = 0; i < strlen(str); i++)
    {
        *(p_str + i) = ' ';
    }

    cout << p_str << endl;

    return 0;
}
```

# Con trỏ void

- Con trỏ void có thể được coi là con trỏ tổng quát, là một kiểu dữ liệu đặc biệt của con trỏ. Con trỏ void có thể trỏ đến bất kỳ đối tượng nào (với bất kỳ kiểu dữ liệu nào) có địa chỉ cụ thể trên bộ nhớ ảo.

```
void *ptr;  
  
int iValue;  
float fValue;  
double dValue;  
string str;  
int iArr[10];  
  
ptr = &iValue;  
ptr = &fValue;  
ptr = &dValue;  
ptr = &str;  
ptr = iArr;
```



# Con trỏ void

- Chúng ta cũng có thể trỏ con trỏ void đến những con trỏ khác.

```
void *ptr;  
  
int *iArr = new int[10];  
ptr = iArr;  
  
delete[] iArr;
```

- Con trỏ void không thể xác định số byte vùng nhớ mà nó trỏ đến. Cần ép kiểu dữ liệu.

```
int value = 5;  
void *vPtr = &value;  
  
int *iPtr = static_cast<int *> (vPtr);  
cout << *iPtr << endl;
```

```
int value = 5;  
void *vPtr = &value;
```

Ép kiểu không đúng gây ra kết quả sai!

```
int *iPtr = static_cast<int *> (vPtr);  
cout << *iPtr << endl;  
  
int64_t * i64Ptr = static_cast<int64_t *> (vPtr);  
cout << *i64Ptr << endl;
```

# Mục đích sử dụng con trỏ void

- Chúng ta thường sử dụng con trỏ void khi mà dữ liệu bên trong vùng nhớ đó không quan trọng. Ví dụ muốn copy dữ liệu từ vùng nhớ này sang vùng nhớ khác mà không quan tâm định dạng của chúng.
- Con trỏ void cũng thường được sử dụng làm tham số của hàm khi muốn input của hàm là con trỏ có kiểu dữ liệu bất kỳ.
- **Chú ý**: Giải phóng vùng nhớ con trỏ void có thể gây ra lỗi vì không biết được kích thước là bao nhiêu. Nên tránh dùng con trỏ void, trừ khi cần thiết.



# Con trỏ trỏ đến con trỏ

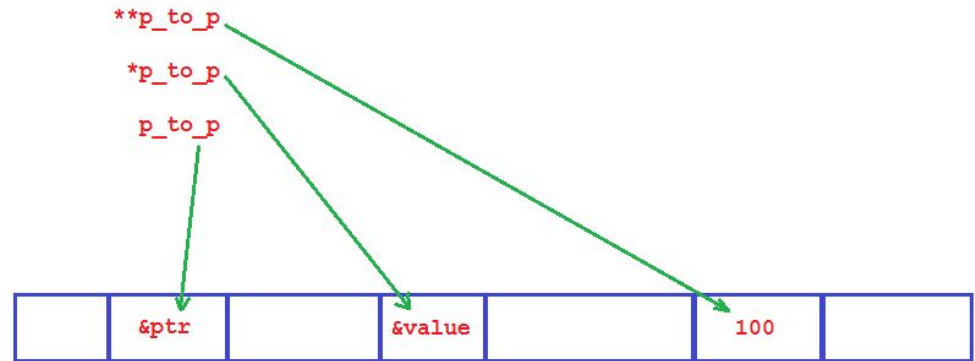
- **Pointer to pointer** là loại con trỏ dùng để lưu trữ địa chỉ của biến con trỏ.
- Khi chúng ta muốn con trỏ trỏ đến con trỏ, đầu tiên thử xem kiểu dữ liệu địa chỉ của con trỏ sẽ là gì?

```
int *ptr = NULL;  
cout << typeid(&ptr).name() << endl;
```



# Con trỏ trỏ đến con trỏ

```
int main()    {  
  
    int value = 100;  
    int *ptr = &value;  
    int **p_to_p = &ptr;  
  
    cout << p_to_p << endl; //print address of ptr  
    cout << *p_to_p << endl; //print address which hold by ptr  
    cout << **p_to_p << endl; //print value at address which hold by ptr  
  
    return 0;  
}
```



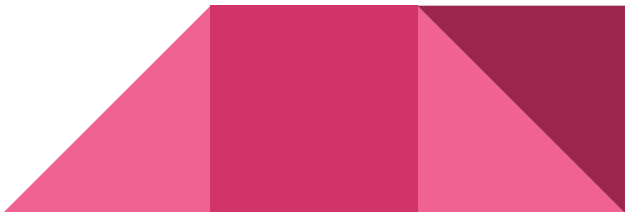
# Con trỏ trỏ đến con trỏ

- Chúng ta không thể khai báo như sau:

```
int **p_to_p = &&value; //not valid
```

- Như con trỏ bình thường, chúng ta cũng có thể khởi tạo pointer to pointer là NULL:

```
int **p_to_p = NULL;
```



# Con trỏ trỏ đến con trỏ

- Pointer to pointer dùng để quản lý mảng 1 chiều kiểu con trỏ:

```
int main()    {  
  
    int *p1 = NULL;  
    int *p2 = NULL;  
    int *p3 = NULL;  
  
    int *p[] = { p1, p2, p3 };  
  
    int **p_to_p = p;  
  
    return 0;  
}
```

```
int *p1 = NULL;  
int *p2 = NULL;  
int *p3 = NULL;  
  
int **p_to_p = new int*[3];  
p_to_p[0] = p1;  
p_to_p[1] = p2;  
p_to_p[2] = p3;  
  
delete[] p_to_p;
```

- Con trỏ kiểu `int *` dùng để trỏ tới các phần tử kiểu `int`, con trỏ kiểu `int **` dùng để trỏ tới các phần tử kiểu `int *`

# Con trỏ trỏ đến con trỏ

- Pointer to pointer dùng để quản lý mảng 2 chiều:

```
int *pToArrPtr[3];  
  
for(int i = 0; i < 3; i++)  
{  
    pToArrPtr[i] = new int[5];  
}
```

3 con trỏ **pToArrPtr** vẫn là được cấp phát trên **stack**. Ta cần cấp phát tất cả trên **Heap**.



# Con trỏ trỏ đến con trỏ

- Cấp phát vùng nhớ Heap trên cho con trỏ 2 chiều:

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

int main()    {
    int **pToArrPtr;

    //Cấp phát vùng nhớ cho 3 con trỏ kiểu (int *)
    pToArrPtr = new int*[3];

    //Mỗi con trỏ kiểu (int *) sẽ quản lý 5 phần tử kiểu int
    for (int i = 0; i < 3; i++)
    {
        pToArrPtr[i] = new int[5];
    }

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            cin >> pToArrPtr[i][j];
        }
    }

    cout << "-----" << endl; }
```

```
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 5; j++)
    {
        cout << pToArrPtr[i][j] << " ";
    }
    cout << endl;
}

//Giải phóng vùng nhớ cho từng dãy vùng nhớ mà 3 con trỏ đang quản lý
for (int i = 0; i < 3; i++)
{
    delete[] pToArrPtr[i];
}

//Giải phóng cho 3 biến con trỏ chịu sự quản lý của pToArrPtr
delete[] pToArrPtr;

return 0;
```



# Con trỏ và hàm

- Chúng ta đã biết hàm có thể nhận tham số **kiểu giá trị** và tham số **kiểu tham chiếu**. Chúng ta còn có thể truyền 1 kiểu tham số nữa vào hàm là **tham số kiểu địa chỉ**, do đó tham số hàm nhận giá trị địa chỉ phải là con trỏ. Chúng ta có thể thay đổi giá trị cho biến được truyền địa chỉ vào.

```
void foo(int *iPtr)
{
    cout << "Int value at " << iPtr << " is " << *iPtr << endl;
}

int main()
{
    int iValue = 10;
    foo(&iValue);

    system("pause");
    return 0;
}
```

```
void printArray(int *arr, int length)
{
    for (int i = 0; i < length; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main()
{
    int iArr[] = { 3, 2, 5, 1, 7, 10, 32 };
    printArray(iArr, sizeof(iArr) / sizeof(int));

    system("pause");
    return 0;
}
```

# Bài tập

- Viết hàm hoán đổi giá trị 2 biến nguyên sử dụng truyền địa chỉ của biến vào tham số hàm.



# Con trỏ và hàm

- Sử dụng pointer to const để làm tham số cho hàm. Đảm bảo giá trị các vùng nhớ được truyền vào hàm sẽ không thay đổi.

```
void printArray(const int *arr, int length)
{
    for (int i = 0; i < length; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main()
{
    int arr[] = {};
    int length = sizeof(arr) / sizeof(int);

    printArray(arr, length);

    system("pause");
    return 0;
}
```

# Con trỏ và hàm

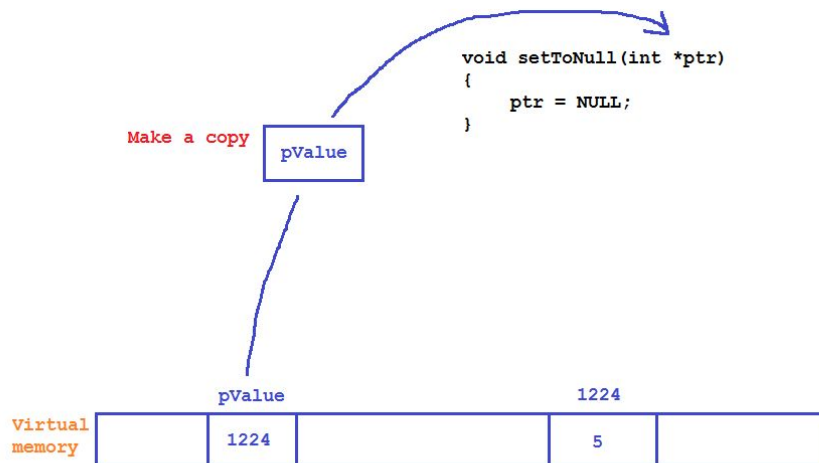
- Tham số hàm có thể là tham chiếu vào con trỏ. Bản chất truyền địa chỉ vẫn như copy giá trị địa chỉ gán cho con trỏ :)

```
void setToNull(int *ptr)
{
    ptr = NULL; // (4)
} // (5)

int main()
{
    int value = 5;
    int *pValue = &value; // (1)

    cout << "pValue point to " << pValue << endl; // (2)
    setToNull(pValue); // (3)
    cout << "pValue point to " << pValue << endl; // (6)

    system("pause");
    return 0;
}
```



# Con trỏ và hàm

- Trong trường hợp chúng ta muốn thay đổi địa chỉ mà con trỏ đang trỏ đến thì sử dụng tham chiếu.

```
void setToNull(int *&ptr)
{
    ptr = NULL;
}

int main()
{
    int value = 5;
    int *pValue = &value;

    cout << "pValue point to " << pValue << endl;
    setToNull(pValue);
    if(pValue == NULL)
        cout << "pValue point to NULL" << endl;
    else
        cout << "pValue point to " << pValue << endl;

    return 0;
}
```

# Con trỏ và hàm

- Hàm có thể trả về kiểu địa chỉ.

```
int * createAnInteger(int value = 0)
{
    int myInt = value;
    return &myInt;
}

int main()
{
    int *pInt = createAnInteger(10);
    cout << *pInt << endl;

    return 0;
}
```

← Đây là 1 lỗi nghiêm trọng với người mới học về con trỏ??  
Tại sao???

# Con trỏ và hàm

- Vùng nhớ stack sẽ bị thu hồi tự động. Để giải quyết vấn đề này, ta cần dùng vùng nhớ Heap.

```
int * createAnInteger(int value = 0)
{
    return new int(value);
}

int main()
{
    int *pInt = createAnInteger(10);

    cout << "Print immediately:  " << *pInt << endl;
    _sleep(5000);
    cout << "After a few seconds: " << *pInt << endl;

    system("pause");
    return 0;
}
```

# Bài tập

- Vấn đề đoạn chương trình này là gì?

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

int * createAnInteger(int value = 0)
{
    return new int(value);
}

int main()
{
    int *pInt = createAnInteger(10);

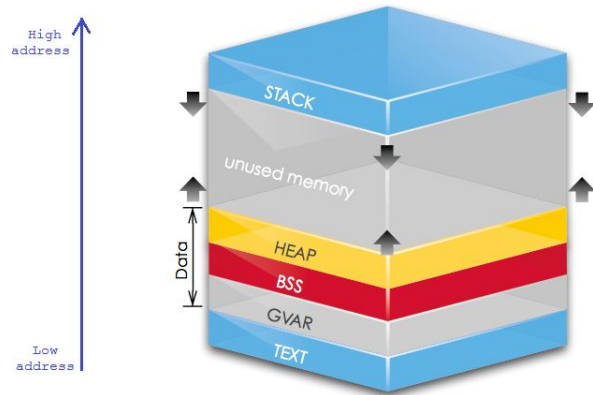
    cout << "Print immediately:  " << *pInt << endl;
    _sleep(5000);
    cout << "After a few seconds: " << *pInt << endl;

    system("pause");
    return 0;
}
```



# Con trỏ hàm

- Mã nguồn của chương trình được load lên RAM lưu ở phần vùng text (phần vùng code segment).



- Sử dụng con trỏ để trỏ đến hàm.

# Con trỏ hàm

- Để in ra địa chỉ 1 hàm, chúng ta làm như sau:

```
int foo()
{
    return 0;
}

int main()
{
    cout << foo << endl;

    return 0;
}
```

- Cú pháp con trỏ hàm:

```
<return_type> (*<name_of_pointer>)( <data_type_of_parameters> );
```

```
int (*pFoo) ();
```

# Con trỏ hàm

- Gán địa chỉ của hàm cho con trỏ hàm:

```
void swapValue(int &value1, int &value2)
{
    int temp = value1;
    value1 = value2;
    value2 = temp;
}

int main()
{
    void(*pSwap) (int &, int &) = swapValue;
    cout << pSwap << endl;
    cout << swapValue << endl;

    return 0;
}
```

- **Chú ý**: Khi lấy địa chỉ hàm, chúng ta gọi duy nhất tên hàm, không thêm ngoặc ().

Con trỏ hàm phải khai báo kiểu dữ liệu và tham số phù hợp mới trỏ đến hàm được.

# Con trỏ hàm

- Sau khi được trỏ đến hàm, con trỏ hàm có thể được sử dụng như hàm thông qua toán tử \*

- **Chú ý**: Tham số mặc định của hàm không

áp dụng cho con trỏ hàm được, vì tham số

mặc định được compiler xác định tại thời

điểm compile chương trình, còn con trỏ

được sử dụng tại thời điểm chương trình

đang chạy.

```
void swapValue(int &value1, int &value2)
{
    int temp = value1;
    value1 = value2;
    value2 = temp;
}

int main()
{
    void(*pSwap) (int &, int &) = swapValue;

    int a = 1, b = 5;
    cout << "Before: " << a << " " << b << endl;
    (*pSwap)(a, b);
    cout << "After: " << a << " " << b << endl;

    return 0;
}
```

# Sử dụng con trỏ hàm làm tham số

- Ví dụ về sắp xếp tăng dần và giảm dần:

- Cần tham số là hàm truyền vào đại diện cho so sánh lớn hơn và nhỏ hơn.
- Sử dụng tham số hàm sắp xếp là một con trỏ hàm.

- Việc sử dụng con trỏ hàm làm tham số giúp thiết kế chương trình đa dạng được hơn (Có thể thiết kế thêm các hàm so sánh khác).

- Con trỏ hàm là tham số cũng có thể khai báo dưới dạng tham số mặc định.



# std::function trong C++11

- Chuẩn C++11 giúp thay thế việc sử dụng con trỏ hàm bằng cách dùng kiểu dữ liệu **function** trong thư viện **functional**.

Cú pháp khai báo biến kiểu `std::function` như sau:

```
std::function< <return_type>([list of parameters]) > varName;
```

Ví dụ:

```
std::function< bool(int, int) > comparisonFunc;
```



# Ví dụ std::function trong C++11

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

void addOne(int &value)
{
    value++;
}

int main()
{
    function<void(int &)> func = addOne;

    int number = 5;
    func(number);

    cout << "New value: " << number << endl;

    return 0;
}
```

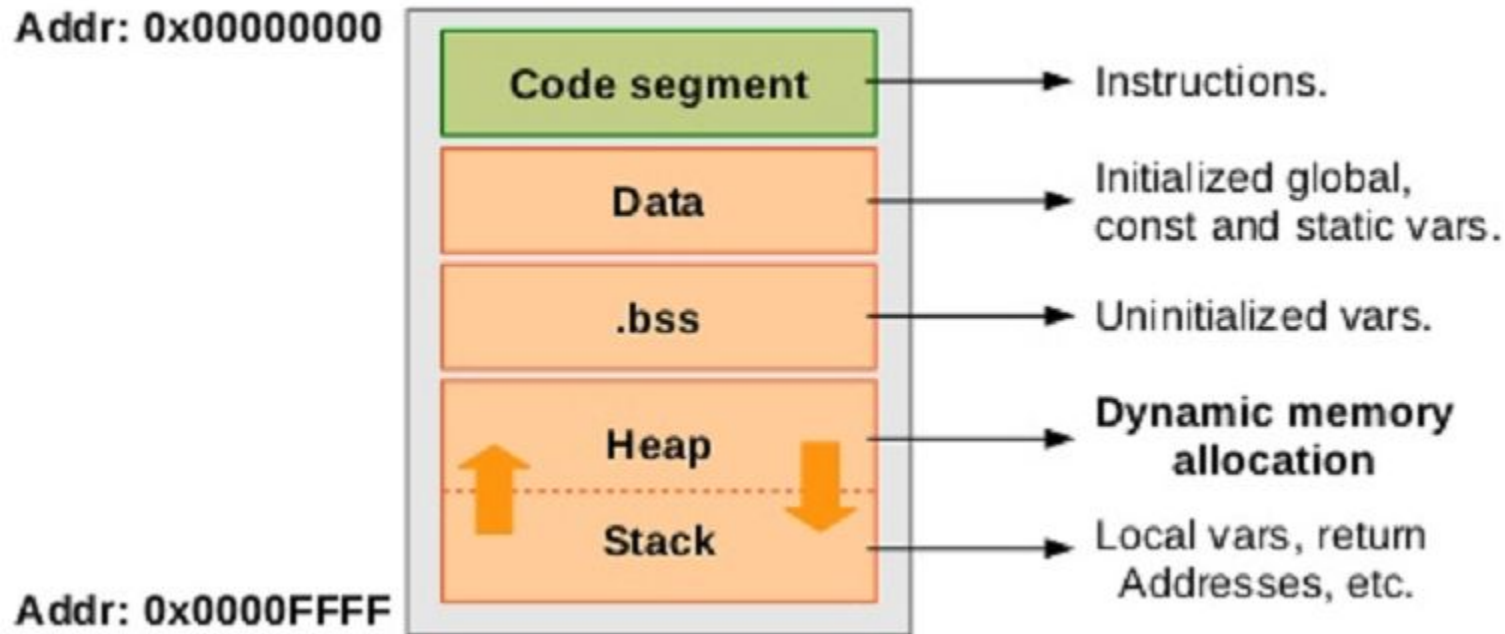
# Bài tập

- Sử dụng con trỏ hàm và `std::function` để tạo hàm có thể thực hiện 4 phép tính cơ bản  $+$ ,  $-$ ,  $*$ ,  $/$ . Biết rằng mỗi phép toán chúng ta cần có một hàm có kiểu trả về float, mỗi hàm có 2 tham số float.






# Các phân vùng bộ nhớ ảo



# Các phân vùng bộ nhớ ảo

- **Code segment:** Lưu trữ mã lệnh đã được biên dịch của chương trình máy tính,
  - **Data segment:** Phân vùng khởi tạo giá trị cho biến static, biến toàn cục.
  - **BSS segment:** Phân vùng khởi tạo biến static, toàn cục, nhưng chưa khởi tạo giá trị cụ thể.
  - **Heap segment:** Phân vùng cấp phát bộ nhớ động.
  - **Stack segment:** Phân vùng cấp phát bộ nhớ cho tham số các hàm, biến cục bộ.
- 

# Một số lỗi thường gặp khi dùng con trỏ

- Con trỏ trỏ đến vùng nhớ ngoài phạm vi:

```
int main()
{
    //allocate memory on Heap
    int *p = new int;

    //p point to somewhere
    p -= 10000;

    //dereference to the area of other program
    *p = 1;

    system("pause");
    return 0;
}
```

```
int main()
{
    int *p = new int[10];
    cout << p << endl;

    delete[] p;

    cout << p << endl;

    _sleep(10000);

    for (int i = 0; i < 10; i++)
        cin >> p[i];

    return 0;
}
```

# Một số lỗi thường gặp khi dùng con trỏ

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

char * getName(char *fullname) {

    if (fullName == NULL)
        return NULL;

    char *pTemp = strrchr(fullname, ' ');

    if (pTemp == NULL)
        return fullname;
    else
        return pTemp + 1;
}
```

```
int main()    {

    char *fullName = new char[50];

    cout << "Enter your full name: ";
    cin.getline(fullName, 50);

    cout << "Your last name is: ";
    char *name = getName(fullName);

    delete[] fullName;

    cout << name << endl;

    return 0;
}
```

# Một số lỗi thường gặp khi dùng con trỏ

- Thử chương trình sau:

```
int main()    {  
  
    char *fullName = new char[50];  
  
    cout << "Enter your full name: ";  
    cin.getline(fullName, 50);  
  
    cout << "Your last name is: ";  
    char *name = getName(fullName);  
  
    delete[] fullName;  
  
    _sleep(5000);  
    cout << name << endl;  
  
    return 0;  
}
```

# Một số lỗi thường gặp khi dùng con trỏ

- Không nên sử dụng toán tử delete cho con trỏ name chương trình trên, sửa lỗi:

```
int main()    {

    char *fullName = new char[50];

    cout << "Enter your full name: ";
    cin.getline(fullName, 50);

    cout << "Your last name is: ";
    char *name = getName(fullName);
    cout << name << endl;

    delete[] fullName;
    fullName = NULL;

    return 0;

}
```

```
int main()    {

    char *fullName = new char[50];

    cout << "Enter your full name: ";
    cin.getline(fullName, 50);

    cout << "Your last name is: ";
    char *name = getName(fullName);
    cout << name << endl;

    name = NULL;

    //keep using fullName
    //and then deallocate it

    delete[] fullName;

    return 0;

}
```

# Một số lỗi thường gặp khi dùng con trỏ

- Con trỏ trỏ đến biến không còn chịu sự quản lý của chương trình:

```
int * newIntValue(int value = 0)
{
    int n = value;
    return &n;
}

int main()    {
    int *pInt = newIntValue(0);

    return 0;
}
```

# Một số lỗi thường gặp khi dùng con trỏ

- Sửa lỗi: Sử dụng cấp phát bộ nhớ động thay vì dùng vùng nhớ Heap

```
int * newIntValue(int value = 0)
{
    return new int(value);
}

int main()
{
    int *p = newIntValue(0);

    delete p;

    return 0;
}
```



# Một số lỗi thường gặp khi dùng con trỏ

- Hiện tượng **memory leak**: là trường hợp cấp phát vùng nhớ cho chương trình (thường trên heap) nhưng vùng nhớ không được sử dụng hoặc không được giải phóng.

- Ví dụ:

```
int *ptr = new int[10];  
//.....  
ptr = NULL;
```

- Khắc phục: Cần có con trỏ thay thế vị trí con trỏ **ptr** trước khi nó trở đi nơi khác.

```
int *ptr = new int[10];  
//.....  
  
int *pTemp = ptr;  
ptr = NULL;
```

# Một số lỗi thường gặp khi dùng con trỏ

```
void resizeArray(int *&p, int oldLength, int newLength)
{
    int *pTemp = p;
    p = allocateArray(newLength);

    //copy data
    if(oldLength < newLength)
    {
        for(int i = 0; i < oldLength; i++)
        {
            p[i] = pTemp[i];
        }
    }
    else
    {
        for(int i = 0; i < newLength; i++)
        {
            p[i] = pTemp[i];
        }
    }
}
```

```
int main()    {

    int length = 10;
    int *p = new int[length];

    int newLength = 20;
    resizeArray(p, length, newLength);

    return 0;
}
```

# Một số lỗi thường gặp khi dùng con trỏ

- Khắc phục:

```
void resizeArray(int *&p, int oldLength, int newLength)
{
    int *pTemp = p;
    p = allocateArray(newLength);

    //copy data
    if(oldLength < newLength)
    {
        for(int i = 0; i < oldLength; i++)
        {
            p[i] = pTemp[i];
        }
    }
    else
    {
        for(int i = 0; i < newLength; i++)
        {
            p[i] = pTemp[i];
        }
    }

    delete[] pTemp;
}
```

```
int main()    {

    int length = 10;
    int *p = new int[length];

    int newLength = 20;
    resizeArray(p, length, newLength);

    delete[] p;

    return 0;
}
```

# Một số lỗi thường gặp khi sử dụng con trỏ

- Sử dụng sai toán tử delete:

```
int main()    {  
    int *p = new int[10];  
    delete p;  
    return 0;  
}
```



# Kiến thức cần nhớ

1. Con trỏ
2. Toán tử trong con trỏ
3. Con trỏ và mảng 1 chiều
4. Con trỏ và mảng ký tự
5. Cấp phát bộ nhớ động
6. Các loại con trỏ hằng
7. Con trỏ void
8. Con trỏ trỏ đến con trỏ
9. Con trỏ và hàm
10. Con trỏ hàm





Thank you!