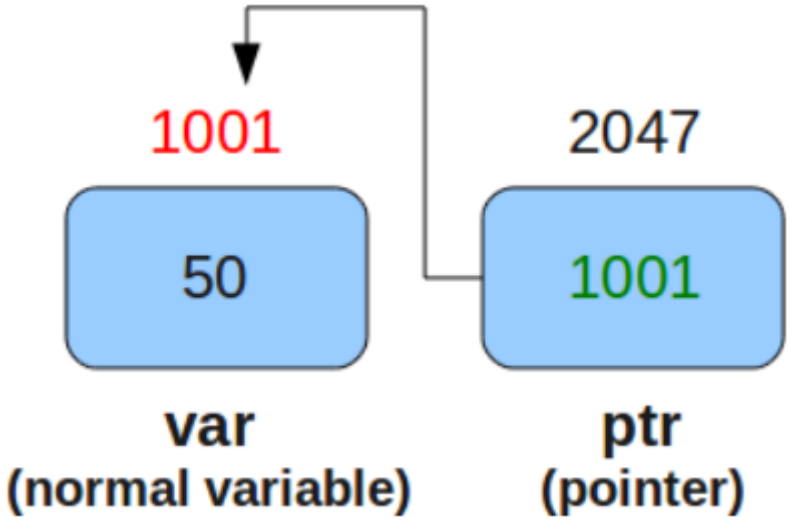
I. Pointer (Con trỏ).

1) Khái niệm.

- Khái niệm: là kiểu dữ liệu dùng để lưu trữ địa chỉ của các đối tượng khác, đối tượng đó có thể là một biến, một chuỗi hoặc hàm.



- Cú pháp:

  <Kiểu dữ liệu> *<Tên pointer>;

  <Kiểu dữ liệu> *<Tên pointer> = <Địa chỉ của đối tượng>;

Lưu ý: khi khai báo nhiều pointer cùng 1 lúc thì dùng cú pháp sau, **lưu ý vị trí dấu *.**

<Kiểu dữ liệu> *<Tên pointer 1>, *<Tên pointer 2>, *<Tên pointer 3>;

- Để lấy địa chỉ của một biến, ta dùng toán tử **&** theo cú pháp: **&<Tên đối tượng>**
- Để truy xuất tới giá trị của biến mà con trỏ đang chỉ tới, ta dùng toán tử ***** với cú pháp: ***<Tên pointer>**
- Con trỏ chỉ có thể lưu địa chỉ của biến có cùng **<Kiểu dữ liệu>** với nó.

  Ví dụ 1:

```
int a = 9;
int* ptr = &a;
printf("ptr = %d\n", ptr);
printf("*ptr = %d\n", *ptr);
*ptr = *ptr + 1;
printf("*ptr + 1 = %d\n", *ptr);
printf("*(&a) = %d\n", *(&a));
```

  Ví dụ 2:

```
int a = 10, b = 100;
int *aPtr = &a, *bPtr = &b;
printf("a = %d\nb = %d", *aPtr, *bPtr);
```

- Con trỏ void: **void *<Tên pointer>**

- Có thể lưu địa chỉ của biến với kiểu dữ liệu bất kỳ.
- Tuy nhiên, không thể truy xuất giá trị bằng thao tác ***<Tên pointer>** được, mà phải thực hiện thao tác ép kiểu.

Ví dụ;

```
    int a = 10;
    int *aPtr = &a;
    char c = 'a';
    char *cPtr = &c;

    void *vPtr;
    vPtr = aPtr;

//  int m = *vPtr + 1; // Error
    int m = *(int*)vPtr + 1;
    printf("%d\n", m);

    vPtr = cPtr;
    char d = *(char*)vPtr + 1;
    printf("%c", d);
```

2) Các thao tác với pointer.

- Ép kiểu pointer: **(<Kiểu dữ liệu>*)<Tên pointer>**
- Không thể thực hiện phép toán *, /, % pointer.
- Không thể thực hiện phép + giữa 2 pointer.
- Được thực hiện phép - giữa 2 pointer, kết quả của phép tính không phải là pointer mà là khoảng cách giữa 2 pointer.
- Có thể thực hiện phép toán + và – một pointer với một số nguyên. Kết quả trả về sẽ là một pointer.

Lưu ý:

- Giả sử ta có biến pointer **ptr** và một số nguyên **x**. Khi thực hiện phép toán **(ptr + x)** thì tương đương với **ptr + x*(sizeof(<Kiểu dữ liệu>))**.
- Hàm **sizeof(<Kiểu dữ liệu>)** sẽ trả dung lượng bộ nhớ cần tiêu tốn của **<Kiểu dữ liệu>** tương ứng.

- Khai báo biến pointer hằng (hằng pointer): **<Kiểu dữ liệu> * const <Tên pointer>;**

Ví dụ:

```
    int a = 10;
    int * const ptr = &a;

//  ptr++; // Error
    (*ptr)++;

    printf("%d", *ptr);
```

- Khai báo đối tượng của một pointer là hằng:

1. const <Kiểu dữ liệu> * <Tên pointer>;

2. <Kiểu dữ liệu> const * <Tên pointer>;

```
    int a = 10;
    const int *ptr = &a;

    ptr++;
//  (*ptr)++; // Error

    printf("%d", *ptr);
```

3) Pointer và mảng.

Ví dụ:

```
   int a[10] = {};

// a++; //Error
   (*a)++;

   printf("%d", *a);
```

➔ **<Tên mảng>** là một pointer hằng.

➔ Ta có thể truy xuất tới phần tử trong mảng bằng cách thứ 2:

***(<Tên mảng> + <Chỉ số>)**

Ví dụ 1: với mảng 1 chiều.

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

for(int i = 0; i < 10; i++){
    printf("%d ", *(a+i));
}
```

Ví dụ 2: với mảng 2 chiều.

```
int a[3][3] = {
    {1, 2, 3}, // a[0]
    {4, 5, 6}, // a[1]
    {7, 8, 9}  // a[2]
};

printf("%d\n", *a[0]);          // a[0][0]
printf("%d\n", *(a[0] + 1));    // a[0][1]
printf("%d\n", *(a[1] + 2));    // a[1][2]

printf("%d\n", *(*(a)));        // a[0][0]
printf("%d\n", *(*(a) + 1));    // a[0][1]
printf("%d\n", *(*(a + 1) + 2));// a[1][2]
```

Ví dụ 3: mảng pointer.

```
int a = 10, b = 11, c = 12;

int *ptr[3] = {&a, &b, &c};

for(int i = 0; i < 3; i++){
    printf("%d ", *ptr[i]);
}

printf("\n");
for(int i = 0; i < 3; i++){
    printf("%d ", *(*(ptr + i)));
}
```

4) Pointer và hàm.

- Tạo hàm với pointer là tham số:

Ví dụ: hàm cộng số b vào số a.

```c
#include <stdio.h>

void sum(int *a, int *b){
    *a = *a + *b;
}

int main(){
    int a, b;
    scanf("%d%d", &a, &b);

    sum(&a, &b);

    printf("%d", a);
}
```

Bài tập: Tạo hàm đổi giá trị của 2 biến số nguyên bằng cách dùng pointer.

```c
#include <stdio.h>

void swap(int *a, int *b){
    int i = *a;
    *a = *b;
    *b = i;
}

int main(){
    int a, b;
    scanf("%d%d", &a, &b);

    swap(&a, &b);

    printf("a = %d\nb = %d", a, b);

}
```

- Tạo hàm trả về pointer:

<Kiểu dữ liệu> *<Tên hàm>(<Danh sách tham số>) { }

Ví dụ: hàm so sánh 2 số và trả về địa chỉ của số lớn hơn.

```c
#include <stdio.h>

int *compare(int *a, int *b){
    if(*a > *b){
        return a;
    } else{
        return b;
    }
}

int main(){
    int a, b;
    scanf("%d%d", &a, &b);

    printf("%d - %d\n", &a, a);
    printf("%d - %d\n", &b, b);

    printf("%d", *compare(&a, &b));
```

```
    printf("%d", compare(&a, &b));
}
```

- Pointer trỏ đến hàm:
    - Tạo pointer:

<Kiểu dữ liệu> (*<Tên pointer>) (<Danh sách tham số>) { }

    - Gán giá trị cho pointer, vì tên hàm cũng là pointer như tên mảng, nên:

<Tên pointer> = <Tên hàm>;

    - Gọi hàm thông qua pointer:

(*<Tên pointer>) (<Danh sách tham số>)

Ví dụ:

```
#include <stdio.h>

void sum(int *a, int *b){
    *a = *a + *b;
}

int main(){
    int a, b;
    scanf("%d%d", &a, &b);

    void (*fPtr) (int*, int*);
    fPtr = sum;
    (*fPtr) (&a, &b);

    printf("%d", a);
}
```

Bài tập: Tạo hàm đổi giá trị của 2 biến số nguyên bằng cách dùng pointer và sử dụng hàm thông qua pointer.

```
#include <stdio.h>

int *compare(int* a, int *b){
    if(*a > *b){
        return a;
    } else{
        return b;
    }
}

int main(){
    int a, b;
    scanf("%d%d", &a, &b);

    int* (*fPtr) (int*, int*);
    fPtr = compare;

    printf("%d", *(*fPtr) (&a, &b));

}
```