# KỸ THUẬT LẬP TRÌNH C

# Chương 10: Con trỏ

1. Các thuộc tính của biến
2. Biến con trỏ
3. Khai báo biến con trỏ
4. Các thao tác trên con trỏ
5. Con trỏ kiểu void
6. Con trỏ kép
7. Con trỏ và mảng
8. Con trỏ và chuỗi
9. Cấp phát vùng nhớ động

# 1. Các thuộc tính của biến

Khi một biến được khai báo có ba thuộc tính sau đây được liên kết với nó, đó là:

- Tên định danh của biến.
- Kiểu dữ liệu của biến.
- Địa chỉ vùng nhớ cấp phát cho biến.

# 2. Biến con trỏ

- Con trỏ là một kiểu dữ liệu dùng để chứa địa chỉ.
- Biến con trỏ là loại biến được dùng để chứa địa chỉ của một biến khác (hay còn gọi là trỏ tới một biến).

Ví dụ: Xét a là một biến thông thường và pa là một biến con trỏ.

Giả sử a có giá trị 500 được lưu trữ tại địa chỉ 1000 trong bộ nhớ và pa được được lưu tại địa chỉ 1008.

Nếu pa đang trỏ tới biến a thì giá trị của pa sẽ là 1000, đó chính là địa chỉ của biến a.

# 3. Khai báo biến con trỏ

**Cú pháp:**

<Kiểu> * <Tên biến con trỏ>;

**ý nghĩa:**

Khai báo một biến có tên là <Tên biến con trỏ> dùng để chứa địa chỉ của các biến có kiểu <Kiểu>.

# Ví dụ:

int *pa; //Khai báo biến con trỏ pa kiểu int

float *pf; //khai báo biến con trỏ pf kiểu float

char *pc; //khai báo biến con trỏ pc kiểu char

void *p; //khai báo biến con trỏ p kiểu void

# 4. Các thao tác trên biến con trỏ

## a. Gán địa chỉ của biến cho biến con trỏ

**Cú pháp:**

<Tên biến con trỏ> = &<Tên biến>;

**Ý nghĩa:**

Gán địa chỉ của biến <Tên biến> cho con trỏ <Tên biến con trỏ>.

# Ví dụ:

```
int a, b;
int *pa, *pb;
pa=&a; // Gán địa chỉ của a cho con trỏ pa
pb=&b; // Gán địa chỉ của b cho con trỏ pb
```

## b. Truy cập nội dung ô nhớ có con trỏ chỉ tới

**Cú pháp:**

*<Tên biến con trỏ>

**ý nghĩa:**

*<Tên biến con trỏ> là nội dung ô nhớ có con trỏ <Tên biến con trỏ> chỉ tới.

# Ví dụ:

```
void main() {
        int a, b, *pa, *pb;
        a = 2;
        b = 3;
        printf("a = %d, b=%d ", a ,b);
        pa=&a; //Gán địa chỉ biến a cho con trỏ pa
        pb=&b; //Gán địa chỉ biến b cho con trỏ pb
        printf("\nNoi dung o nho co pa tro toi = %d\n", *pa);
        printf("\nNoi dung o nho co pb tro toi = %d \n", *pb);
        *pa=20; // Thay đổi giá trị của *pa
        *pb=30; // Thay đổi giá trị của *pb
        printf("a = %d, b = %d\n", a, b); // a, b thay đổi theo
}
```

## c. Một số phép toán trên con trỏ

- **Phép gán con trỏ**:

Hai con trỏ cùng kiểu có thể gán cho nhau.

Ví dụ:

```
void main() {
    int a = 10;
    int *p, *q;
    p = &a;
    printf("%d\n", *p);
    q = p; //q và p cùng trỏ tới biến a
    printf("%d\n", *q);
}
```

- **Phép chuyển kiểu con trỏ**

  **Cú pháp:**

  (<Kiểu>*) <Tên biến con trỏ>

  **Ý nghĩa:**

  Chuyển kiểu của <Tên biến con trỏ> thành <Kiểu>*

# Ví dụ:

```
void main() {
        int a = 10;
        int *p, *q;
        float *f;
        p = &a;
        q = p;
        f = (float*)p; //Chuyển kiểu con trỏ p thành kiểu float*
        printf("%d\n", *((int *)f));
}
```

- **Cộng, trừ con trỏ với một số nguyên**

  Ta có thể cộng (+), trừ (-) một con trỏ với một số nguyên n nào đó, kết quả trả về là một con trỏ mới. Con trỏ này trỏ đến ô nhớ cách ô nhớ của con trỏ hiện tại n phần tử theo chiều tăng hoặc giảm.

# Ví dụ:

```
void main() {
    int a[10] = {10, 40, 70, 90, 30, 80, 60, 50, 20, 0};
    int *p1, *p2, *p3;
    p1 = &a[0];
    p2 = p1 + 7; /*p2 trỏ đến ô nhớ cách ô nhớ của con
            trỏ p1 bảy phần tử theo chiều tăng*/
    p3 = p2 - 3; /*p3 trỏ đến ô nhớ cách ô nhớ của con
            trỏ p2 ba phần tử theo chiều giảm*/
    printf("%d, %d, %d\n" , *p1, *p2, *p3) ;
}
```

- **So sánh hai con trỏ**

Hai biến con trỏ cùng kiểu có thể được so sánh trong một biểu thức quan hệ (==, !=, >, >=, <, <=)

# Ví dụ:

```
void main() {
        int a = 10, b = 20;
        int *pa = &a;
        int *pb = &b;
        if(pa > pb)
                printf("Dia chi bien a > dia chi bien b\n");
        else if(pa < pb)
                printf("Dia chi bien a < dia chi bien b\n");
        else  printf("Dia chi bien a bang dia chi bien b\n");
}
```

# 5. Con trỏ kiểu void

Con trỏ kiểu void là một loại con trỏ có thể trỏ đến các biến có kiểu bất kỳ.

**Cú pháp khai báo:**

void *<tên biến con trỏ>;

# Ví dụ:

```
void main() {
        void *p; //p là con trỏ kiểu void
        char c = 'a';
        int d = 10;
        float f = 30.50;
        p = &c;
        printf("%c\n" , *((char *)p)); /*Phải ép kiểu vì p là con
                                          trỏ kiểu void*/
        p = &d;
        printf("%d\n", *((int *)p));
        p = &f;
        printf("%0.2f\n", *((float *)p));
}
```

# 6. Con trỏ kép

Con trỏ kép là một loại con trỏ được dùng để chứa địa chỉ của một con trỏ khác.

**Cú pháp khai báo:**

<Kiểu> ** <Tên biến con trỏ>;

## Ví dụ:

```
void main() {
        char *p[3] = {“One”, “Two”, “Three”}; /*Mảng 3 con
                                        trỏ chỉ đến 3 chuỗi*/
        char **pp;      //pp là một con trỏ kép
        pp = p;         //pp chứa địa chỉ của con trỏ p[0]
        printf(“%s\n%s\n%s\n”, *pp, *(pp+1), *(pp+2));
}
```

# 7. Con trỏ và mảng

## a. Con trỏ và mảng một chiều

- **Tên mảng dùng như con trỏ**

Tên mảng thực chất là hằng địa chỉ, nó chính là địa chỉ phần tử đầu tiên của mảng.

Giả sử a là tên mảng, ta có quy tắc sau:

**a** tương đương với **&a[0]**

**a + i** tương đương với **&a[i]**

***(a + i)** tương đương với **a[i]**

# Ví dụ:

```
void main() {
        int a[SIZE];
        int n; int i;
        do {
                printf("Nhap so phan tu:"); scanf("%d", &n);
        } while(n < 1 || n > SIZE);
        for(i = 0; i < n; i++) {
                printf("pt thu %d:", i);
                scanf("%d", a + i);
        }
        for(i = 0; i < n; i++)
                printf("%d\t", *(a + i));
        printf("\n");
}
```

- **Con trỏ chỉ đến phần tử mảng**

  Xét khai báo sau:

  int a[SIZE];

  int *p;

  Nếu có câu lệnh p = a thì

  **p trỏ tới a[0]**

  **p + 1 trỏ tới a[1]**

  **. . .**

  **p + i trỏ tới a[i]**

# Ví dụ:

```
void main() {
        int a[SIZE];
        int n;
        int i, *p;
        p = a;
        do {
                printf("Nhap so phan tu:"); scanf("%d", &n);
        } while(n < 1 || n > SIZE);
        for(i = 0; i < n; i++) {
                printf("pt thu %d:", i); scanf("%d", p + i);
        }
        for(i = 0; i < n; i++)
                printf("%d\t", *(p + i));
        printf("\n");
}
```

## b. Con trỏ và mảng hai chiều

- **Tên mảng dùng như con trỏ**

Một mảng hai chiều có thể hiểu như là mảng (một chiều) của mảng, tức là mỗi phần tử của mảng lại là mảng một chiều.

Giả sử a là tên mảng 2 chiều, ta có quy tắc:

**a[i] tương đương với &a[i][0]**

**a[i] + j tương đương với &a[i][j]**

***(a[i] + j) tương đương với a[i][j]**

**Hoặc**

***(a + i) + j tương đương với &a[i] [j]**

***(*(a + i) + j) tương đương với a[i] [j]**

# Ví dụ:

```
void main() {
        int a[SIZE1][SIZE2]; int sd, sc; int i, j;
        do {
                printf("So dong va so cot:");
                scanf("%d%d", &sd, &sc);
        } while(sd < 1 || sd > SIZE1 || sc < 1 || sc > SIZE2);
        for(i = 0; i < sd; i++)
                for(j = 0; j < sc; j++)  {
                printf("pt thu [%d][%d]:", i, j);
                scanf("%d", *(a + i) + j); }
        for(i = 0; i < sd; i++)  {
                for(j = 0; j < sc; j++)
                        printf("%d\t", *(*(a + i) + j));
                printf("\n");}
}
```

- **Con trỏ chỉ đến phần tử mảng**

  Xét khai báo sau :

  int a[SIZE1][SIZE2] ;

  int (*p)[SIZE2]; /*Khai báo p là một con trỏ kiểu int[SIZE2] */

  Nếu có câu lệnh **p = a** thì

  ***(p + i) + j tương đương với &a[i] [j]**

  ***(*(p + i) + j) tương đương với a[i] [j]**

Ví dụ:

```
void main() {
        int a[SIZE1][SIZE2]; int i, j, sd, sc;
        int (*p)[SIZE2]; p = a;
        do {
                printf("so dong va so cot:");
                scanf("%d%d", &sd, &sc);
        } while(sd < 1 || sd > SIZE1 || sc < 1 || sc > SIZE2);
        for(i = 0; i < sd; i++)
                for(j = 0; j < sc; j++)  {
                        printf("pt thu [%d][%d]:", i, j);
                        scanf("%d", *(p + i) + j);}
        for(i = 0; i < sd; i++)  {
                for(j = 0; j < sc; j++)
                        printf("%d\t", *(*(p + i) + j));
                printf("\n"); }}
```

# 8. Con trỏ và chuỗi

## a. Khai báo

Để khai báo s là một chuỗi ký tự và p là con trỏ trỏ đến chuỗi ký tự, ta viết:

```
char s[SIZE];
char *p;
```

Để khởi tạo một chuỗi trong cả hai trường hợp, ta viết:

```
char s[SIZE] = <Hằng chuỗi>;
char *p = <Hằng chuỗi>; /*p trỏ đến
                          đầu chuỗi*/
```

## b. Xét một số ví dụ về hàm xử lý chuỗi

Ví dụ: Viết một hàm stringcpy() có chức năng tương tự như hàm strcpy() có trong thư viện string.h.

```
void stringcpy (char *dest, char *source) {
    while((*dest = *source) != '\0') {
            dest++;
            source++;
    }
}
```

Ví dụ: Viết hàm stringcmp() có chức năng tương tự như hàm strcmp() có trong thư viện string.h.

```
int stringcmp (char *s1, char *s2) {
        while (*s1 == *s2)  {
                if (*s1 == '\0')
                        return 0;        //s1 bằng s2
                s1++;
                s2++;
        }
        return (*s1 - *s2); /*s1 khác s2 (nếu *s1-* s2 > 0
                thì s1 lớn hơn s2, ngược lại s1 nhỏ hơn s2)*/
}
```

## b. Mảng con trỏ chỉ đến chuỗi

- **Khai báo mảng con trỏ chỉ đến chuỗi**

  **Cú pháp:**

  char *<tên mảng>[SIZE];

  **ý nghĩa:**

  Khai báo một mảng gồm SIZE con trỏ chỉ đến chuỗi.

- **Khởi tạo mảng con trỏ chỉ đến chuỗi**

  **Cú pháp:**

  char *<tên mảng>[SIZE] = {<danh sách các hằng chuỗi>};

Ví dụ: Khởi tạo một mảng 5 con trỏ chỉ đến 5 chuỗi tên

```
char *ds[5] = { “Uyen”, “Quan”,”Tri”,
                “Nhung”, “My” };
```

Với cách khai báo mảng con trỏ như trên thì lượng ô nhớ sẽ cấp phát vừa đủ cho từng phần tử của mảng.

# Ví dụ: Sắp xếp danh sách theo chỉ mục

```
#define SI_SO_MAX      50
#define HO_TEN_MAX  30
void main() {
      char ds[SI_SO_MAX][ HO_TEN_MAX];
      char *p[SI_SO_MAX];
      char *tmp;     int i, j, n;
      do {
            printf("Nhap si so lop:");
            scanf("%d", &n);
      } while(n < 1 || n > SI_SO_MAX);
      fflush(stdin);
      for(i = 0; i < n; i++) {
            printf("Hoc vien thu %d:", i);
            gets(ds[i]);
            p[i] = ds[i]; }
```

```
    //Sắp xếp mảng các con trỏ theo chỉ mục
    for(i = 0; i < n - 1; i++)
        for(j = i + 1; j < n; j++) {
            if (strcmp(p[i], p[j]) > 0) {
                tmp = p[i];
                p[i] = p[j];
                p[j] = tmp;
            }
        }
    //Xuất danh sách lớp
    printf("Danh sach lop sau khi sap xep:\n");
    for(i = 0; i < n; i++)
        printf("%s\n", p[i]);
    printf("\n");
}
```

# 9. Cấp phát vùng nhớ động

## a. Biến động (Dynamic variable)

- Biến động là các biến được tạo lúc chạy chương trình.
- Các biến động không có tên và việc truy nhập các biến động được thực hiện thông qua biến con trỏ chỉ đến nó.

## b. Cấp phát vùng nhớ động

**Cú pháp**:

```
void *malloc(size_t size);
void *calloc(size_t nitems, size_t size);
```

**Ý nghĩa:**

Hàm malloc sẽ cấp phát một vùng nhớ động có kích thước là size bytes.

Hàm calloc sẽ cấp phát một vùng nhớ động có kích thước là nitems*size bytes.

Nếu thành công cả hai hàm này đều trả về con trỏ chỉ đến đầu vùng nhớ được cấp phát, ngược lại nếu thất bại sẽ trả về giá trị NULL.

Ví dụ:

```
void main() {
	int n;
	int *p;
	printf("Nhập số phần tử mảng: ");
	scanf("%d", &n);
	p = (int*) malloc(n * sizeof(int));
	Nhap(p, n);    //Nhập mảng
	Xuat(p, n);    //Xuất mảng
	free(p);
}
```

# Ví dụ:

```
void main() {
        int **p, sd, sc, i;
        printf("Nhập số dòng và số cột: ");
        scanf("%d%d", &sd, &sc);
        p = (int**) malloc(sd * sizeof(int *));
        for(i = 0; i < sd; i++)
                p[i] = (int *)malloc(sc * sizeof(int));
        Nhap(p, sd, sc);        //Nhập mảng
        Xuat(p, sd, sc);        //Xuất mảng
        for(i = 0; i < sd; i++)
                free(p[i]);
        free(p);
}
```

## c. Cấp phát lại vùng nhớ

**Cú pháp:**

void *realloc(void *p, size_t size);

**Ý nghĩa:**

Cấp phát lại 1 vùng nhớ do con trỏ p quản lý, vùng nhớ này có kích thước mới là **size** bytes, khi cấp phát lại thì nội dung vùng nhớ trước đó vẫn tồn tại.

Hàm trả về con trỏ chỉ đến đầu vùng nhớ được cấp phát nếu thành công, ngược lại trả về NULL và không giải phóng hay thay đổi vùng nhớ cũ

```
void main() {
	int *ptr, ptam, i;
	ptr = (int *)calloc(5, sizeof(int));
	if(ptr != NULL) {
		*ptr = 1;
		*(ptr + 1) = 2;
		ptr[2] = 4; ptr[3] = 8; ptr[4] = 16;
		ptam = (int*)realloc(ptr, 7 * sizeof(int));
		if(ptam != NULL)  {
			ptr = ptam;
			ptr[5] = 32; ptr[6] = 64;
			for(i = 0; i < 7; i++)
				printf("ptr[%d] : %d\n", i, ptr[i]);
		} else printf("Khong du bo nho - realloc that bai.\n");
		free(ptr);
	} else printf("Khong du bo nho - calloc that bai\n");
}
```

## d. Giải phóng vùng nhớ động

**Cú pháp:**

void free(void *p);

**Ý nghĩa**:

Giải phóng vùng nhớ được quản lý bởi con trỏ p.

# Hết