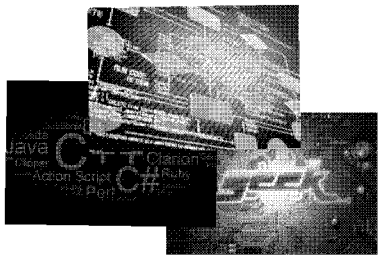
ĐẠI HỌC QUỐC GIA TPHCM
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

NHẬP MÔN LẬP TRÌNH

CHƯƠNG VII

CON TRỎ



Nguyễn Trọng Chỉnh
chinhnt@uit.edu.vn

CON TRỎ

- ĐỊA CHỈ
- KHÁI NIỆM CON TRỎ
- CÁC PHÉP TOÁN
- CON TRỎ VÀ MẢNG
- CẤP PHÁT ĐỘNG
- CON TRỎ HÀM (sinh viên tự đọc)

CON TRỎ

❖ĐỊA CHỈ

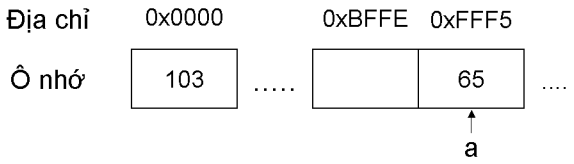
- Bộ nhớ máy tính là một tập hợp các byte (8bit) được đánh dấu bằng một số kiểu int được gọi là địa chỉ (address). Địa chỉ được sử dụng để xác định vị trí ô nhớ thích hợp để tương tác.

- Khi một biến được khai báo, sẽ có một số ô nhớ trên stack được sử dụng để lưu trữ dữ liệu của biến.

Ví dụ:

char a;

a = 65;



3

CON TRỎ

❖ĐỊA CHỈ

- Khi cần thay đổi nội dung dữ liệu của biến, địa chỉ ô nhớ sẽ được dùng để xác định vị trí của nó trên bộ nhớ của chương trình và nội dung mới được ghi vào ô nhớ đó.

- Cú pháp để lấy địa chỉ ô nhớ bắt đầu của một biến:

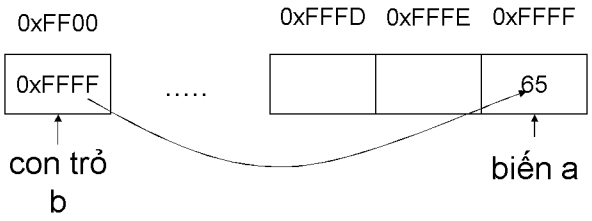
&tên_biến

4

CON TRỎ

❖KHÁI NIỆM CON TRỎ

Con trỏ (pointer) là kiểu dữ liệu lưu trữ địa chỉ vùng nhớ, dùng để xác định và truy cập đến vùng nhớ đó của chương trình.



5

CON TRỎ

❖KHÁI NIỆM CON TRỎ

- Biến con trỏ được khai báo theo cú pháp sau:

kiểu *tên_biến;

- Địa chỉ vùng nhớ là một số kiểu int nên có thể khởi tạo giá trị ban đầu cho con trỏ là một giá trị kiểu int hoặc là địa chỉ vùng nhớ xác định nào đó.

- Giá trị **NULL** (có giá trị 0) là giá trị đặc biệt dùng để gán cho con trỏ chưa được dùng để quản lý một vùng nhớ nào.

6

CON TRỎ

❖KHÁI NIỆM CON TRỎ

- Ví dụ: khai báo biến con trỏ:

```
struct Hocsinh {
    char hoten[31];
     int namsinh;
};
char a = 'A';
int *pInt = NULL;
char *pChr = &a;
struct Hocsinh *hs[30];
```

7

CON TRỎ

❖CÁC PHÉP TOÁN

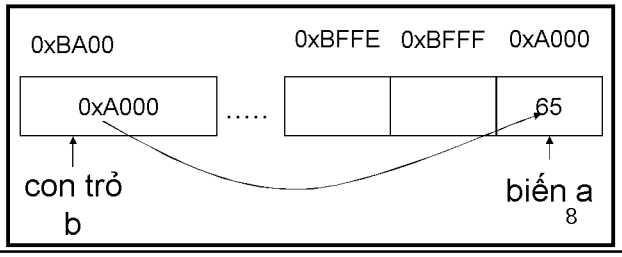
•Phép gán "=": gán địa chỉ một vùng nhớ cho con trỏ.

•Phép phân giải địa chỉ (dereferencing) "*": trả về dữ liệu được chứa tại địa chỉ mà con trỏ lưu trữ. Dữ liệu được xử lý theo kiểu mà con trỏ được khai báo.

Ví dụ:

```
char *b;
char a = 'A';
b = &a;
printf("%c", *b);
```

A



8

CON TRỎ

❖CÁC PHÉP TOÁN

Lưu ý: phân giải địa chỉ cho biến con trỏ kiểu struct để truy cập từng thành phần của biến, sử dụng một trong hai cú pháp sau:

(*tên_biến).thành_phần

tên_biến->thành_phần

Ví dụ:

```
printf((*hs[0]).hoten);
printf("%d", hs[0]->namsinh);
```

9

CON TRỎ

❖CÁC PHÉP TOÁN

•Phép cộng với một số nguyên n "+": tăng giá trị vùng nhớ mà con trỏ đang quản lý lên n lần kích thước kiểu của con trỏ

Ví dụ:

```
int *p = 0;
p = p + 1;
printf("%u", p);
```

2 *(nếu sử dụng hệ điều hành 32bit sẽ là 4)*

10

CON TRỎ

❖CÁC PHÉP TOÁN

•Phép trừ với một số nguyên n "-": giảm giá trị vùng nhớ mà con trỏ đang quản lý xuống n lần kích thước kiểu của con trỏ.

Ví dụ:

```
int *p = (int *)8;
p = p - 2;
printf("%u", p);
```

4 *(nếu sử dụng hệ điều hành 32bit sẽ là 0)*

11

CON TRỎ

❖CÁC PHÉP TOÁN

•Phép toán **++** có kết quả như phép toán **+** với n=1, phép toán ghép **+=** thực hiện phép gán sau khi thực hiện phép **+** với một số nguyên n.

•Các phép toán **--** có kết quả như phép toán **-** với n=1, phép toán ghép **-=** thực hiện phép gán sau khi thực hiện phép **-** với một số nguyên n.

•Lưu ý: nếu con trỏ p chứa địa chỉ ô nhớ chưa được chương trình cấp phát, thì việc ghi dữ liệu vào vùng nhớ mà p đang trỏ sẽ sinh ra lỗi.

12

CON TRỎ

❖CÁC PHÉP TOÁN

Ví dụ: viết chương trình nhập vào một chuỗi và in ra các ký tự in hoa của chuỗi

```
char s[20], *p;
gets(s);
p = s;
while (*p != 0) {
    if ((*p > 64) && (*p < 97)) printf("%c ", *p);
    p++;
}
```

13

CON TRỎ

❖CÁC PHÉP TOÁN

Ví dụ: viết chương trình nhập vào một chuỗi và in ra chuỗi đảo ngược của chuỗi vừa nhập

```
char s[20], *p;
gets(s);
p = s + strlen(s); // cần #include <string.h>
do {
    printf("%c ", *p);
    p--;
} while (p != s - 1)
```

14

CON TRỎ

❖BÀI TẬP

1)cho đoạn chương trình sau, cho biết kết quả:

```
char a = 'A';
char b = 'B';
char *pC = &a;
int *pI = (int *)&pC;
printf("%c", *pC);
*pI = (int)&b;
printf("%c", *pC);
```

15

CON TRỎ

❖BÀI TẬP

2)cho đoạn chương trình sau, cho biết kết quả:

```
void abc(int *a, int *b) {
    *a = *a + *b; *b = *a - *b; *a = *a - *b;
}
void main() {
    int x = 2, y = 8;
    abc(&x, &y);
    printf("%d %d", x, y);
}
```

16

CON TRỎ

❖CON TRỎ VÀ MẢNG

- Mảng thực chất là một hằng con trỏ có giá trị là một địa chỉ trên stack. Khi khai báo một mảng A, chương trình sẽ đánh dấu một vùng nhớ với kích thước bằng với kích thước của A và gán cho A địa chỉ của ô nhớ đầu tiên của vùng này.
- Các phép toán dùng trên mảng đều áp dụng được cho con trỏ.

17

CON TRỎ

❖BÀI TẬP

3) Viết hàm đảo ngược thứ tự một chuỗi ký tự. Viết chương trình sử dụng hàm này để nhập một chuỗi và in ra chuỗi đảo ngược của nó.

4) Viết chương trình nhập vào một bảng A gồm mxn số nguyên (m, n không quá 10) và in ra bảng A đã được sắp xếp theo giá trị tăng dần như sau:

$$A[0,0] \le .. \le A[0, n-1]$$
$$\le A[1, 0] \le .. \le A[1,n-1]$$
$$\le A[m-1,0] \le .. \le A[m-1,n-1]$$

18

CON TRỎ

```
#include <stdio.h>
#include <string.h>
void invert(char *s) {
    char t;
    int i, n, m;
    n = strlen(s); m = n/2;
    for (i = 0; i < m; i++) {
            t = s[i]; s[i] = s[n - i - 1]; s[n - i - 1] = t;
    }
}
```

19

CON TRỎ

```
void main() {
    char st[100];
    gets(st);
    invert(st);
    printf(st);
}
```

20

CON TRỎ

```
#include <stdio.h>
void sort(int *a, int n) {
    int i, j, t;
    for (i = 0; i < n - 1; i++)
            for (j = i + 1; j < n; j++)
                if (a[i] > a[j]) { t = a[i]; a[i] = a[j]; a[j] = t; }
}
```

21

CON TRỎ

```
void nhap(int *a, int *m, int *n) {
    int i, j;
    scanf("%d %d", m, n);
    for (i = 0; i < *m; i++)
            for (j = 0; j < *n; j++)
                scanf("%d", &a[*n * i + j]);
}
```

22

CON TRỎ

```
void xuat(int *a, int m, int n) {
    int i, j;
    for (i = 0; i < m; i++) {
            for (j = 0; j < n; j++)
                printf("%d ", a[n * i + j]);
            printf("\n");
    }
}
void main() {
    int a[10][10], m, n;
    nhap((int *)a, &m, &n); sort((int *)a, m * n); xuat((int *)a, m, n);
}
```

23

CON TRỎ

❖CẤP PHÁT ĐỘNG

- Cấp phát động (dynamic memory allocation) nhằm xác định vùng nhớ để lưu trữ dữ liệu trên bộ nhớ để các vùng dữ liệu của các biến không bị chồng lấn nhau. Việc cấp phát động được thực hiện trên vùng nhớ HEAP. Vùng nhớ được cấp phát sẽ được quản lý bởi một con trỏ.

- Cấp phát động có thể thực hiện bất cứ nơi nào trong chương trình nên việc sử dụng bộ nhớ sẽ hiệu quả hơn.

- Các hàm liên quan đến cấp phát động được khai báo trong file **stdlib.h**

24

CON TRỎ

❖CẤP PHÁT ĐỘNG

- Cú pháp cấp phát động trên C:

+ Cấp phát một biến dùng hàm malloc:

kiểu *biến;

biến = (kiểu *)malloc(số_byte);

+ Cấp phát một mảng dùng hàm calloc:

kiểu *biến;

biến = (kiểu *)calloc(số_pt, sizeof(kiểu));

25

CON TRỎ

❖CẤP PHÁT ĐỘNG

- Cú pháp cấp phát động trên C++ dùng toán tử **new**:

+ Cấp phát một biến:

kiểu *biến;

biến = new kiểu;

+ Cấp phát một mảng:

kiểu *biến;

biến = new kiểu[số_pt];

26

CON TRỎ

❖CẤP PHÁT ĐỘNG

- Nếu chương trình không còn đủ vùng nhớ và cấp phát không thành công, biến con trỏ sẽ nhận giá trị NULL.

Lưu ý: việc thay đổi giá trị vùng nhớ trên một con trỏ có giá trị NULL sẽ sinh lỗi chương trình.

27

CON TRỎ

❖CẤP PHÁT ĐỘNG

- Giải phóng vùng nhớ nhằm thông báo cho chương trình biết vùng nhớ đó có thể sử dụng để lưu trữ dữ liệu cho các biến khác.

- Cú pháp giải phóng vùng nhớ trên C dùng hàm free:

free(biến);

- Cú pháp giải phóng vùng nhớ trên C++ dùng toán tử delete:

+ Đối với biến thường: **delete biến;**

+ Đối với biến mảng: **delete biến[];**

28

CON TRỎ

❖**Bài tập**

5) Viết hàm nhập một dãy số nguyên có kích thước tùy ý và hàm tính tổng các số chẵn của một dãy. Viết chương trình nhập vào một dãy các số nguyên có kích thước tùy ý và in ra tổng các số chẵn trong dãy.

6) Định nghĩa cấu trúc điểm, cấu trúc tam giác, viết các hàm nhập điểm, nhập tam giác, in điểm, in tam giác. Viết chương trình nhập vào một danh sách các tam giác với số lượng tùy ý, in ra thông tin hình chữ nhật có chu vi lớn nhất.

7) Viết chương trình nhập vào một bài thơ với số câu tùy ý và độ dài câu tùy ý, in ra màn hình bài thơ vừa nhập theo thứ tự câu ngược lại với thứ tự nhập.

29

CON TRỎ

```
#include <stdio.h>
#include <stdlib.h>
int *nhap(int *n) {
    int i, *ret;
    scanf("%d", n);
    ret = (int *) malloc(*n * sizeof(int));
    if (ret == NULL)
            return ret;
    for (i = 0; i < *n; i++)
            scanf("%d", &ret[i]);
    return ret;
}
```

30

CON TRỎ

```
int tong(int *a, int n) {
    int s = 0, i;
    for (i = 0; i < n; i++)
            if (a[i] % 2 == 0) s += a[i];
    return s;
}
```

31

CON TRỎ

```
void main() {
    int *a, n;
    a = nhap(&n);
    if (a != NULL) {
            printf("%d\n", tong(a, n);
            free(a);
    }
}
```

32

CON TRỎ

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
typedef struct {
    float x, y;
} Diem;
typedef struct _Tamgiac{
    Diem a, b, c;
    float p, s;
    struct _Tamgiac *next;
}TAMGIAC, *Tamgiac;
```

33

CON TRỎ

```
void nhapdiem(Diem *d) {
    scanf("%f%f", &d->x, &d->y);
}
void indiem(Diem d) {
    printf("x=%.2f, y=%.2f\n", d.x, d.y);
}
float kcach(Diem a, Diem b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y)*(a.y - b.y));
}
```

34

CON TRỎ

```
void nhaptg(Tamgiac t) {
    float a, b, c, p;
    nhapdiem(&t->a); nhapdiem(&t->b); nhapdiem(&t->c);
    a = kcach(t->a, t->b); b = kcach(t->b, t->c);
    c = kcach(t->c, t->a);
    t->p = a + b + c;
    p = t->p / 2;
    t->s = sqrt(p * (p - a) * (p - b) * (p - c));
    t->next = NULL;
}
```

35

CON TRỎ

```
void intg(TAMGIAC t) {
    printf("diem a: "); indiem(t.a);
    printf("diem b: "); indiem(t.b);
    printf("diem c: "); indiem(t.c);
    printf("chu vi: %.2f\n", t.p);
    printf("dien tich: %.2f\n", t.s);
}
```

36

CON TRỎ

```
void main() {
    Tamgiac ds;
    int n, i, max;
    scanf("%d", &n);
    ds = (Tamgiac) malloc(n * sizeof(TAMGIAC));
    if (ds == NULL) return;
        for (i = 0; i < n; i++)
            nhaptg(&ds[i]);
    max = 0;
    for (i = 1; i < n; i++);
            if (ds[max].p < ds[i].p) max = i;
    intg(ds[max]);
    free(ds);
}
```

37