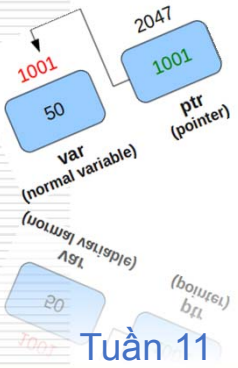
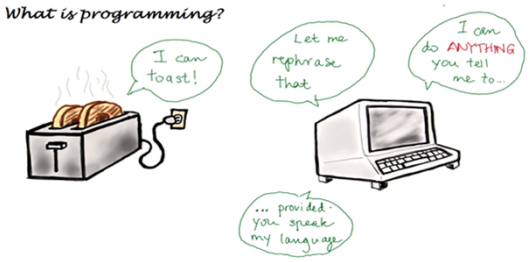




Tuần 11

Con trỏ (Pointer)

CT101 – Lập trình căn bản

Khoa CNTT&TT – ĐHCT

Mục đích

Nhằm giới thiệu khái niệm con trỏ
và các ứng dụng của con trỏ trong lập trình,
minh họa cụ thể bằng ngôn ngữ lập trình C.

Yêu cầu

- Có thể trình bày khái niệm con trỏ
- Biết sử dụng khai báo con trỏ
- Biết sử dụng các phép toán trên con trỏ (*, &)
- Giải thích được tổ chức bộ nhớ của một c/trình
- Có khả năng vận dụng con trỏ để:
  - Tạo mảng động
  - Truyền tham số cho hàm
  - Tạo con trỏ kiểu cấu trúc
- Trình bày được một số ứng dụng nâng cao của con trỏ

Nội dung

1. Tại sao cần có con trỏ?
2. Khai báo và sử dụng con trỏ
3. Ứng dụng của con trỏ
   i. Cấp phát động bộ nhớ & mảng động
   ii. Truyền tham số cho hàm bằng con trỏ
   iii. Con trỏ kiểu cấu trúc
   iv. Một số ứng dụng con trỏ nâng cao (giới thiệu).
   v. Một số lỗi thường gặp
4. Kiểm tra kiến thức và bài tập
5. Phụ lục

"It's considered one of the most frightening topics in all of programming."

"Pointers scare a lot of beginning C programmers – and even experienced programmers of other languages. I believe that the reason for the fear and misunderstanding is that no one bothers to explain in fun, scintillating detail how pointers really work. **So clear your mind, crack your knuckles, and get ready to embrace one of the C language's most unique and powerful features.**"

-- Dan Gookin –

(Beginning programming with C for Dummy – Wiley, 2013)

1. Tại sao cần con trỏ?

Ví dụ 1 – Hàm swap()

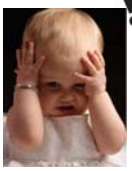
Cho hàm swap() như sau:

```
void swap(int a, int b) {
    int tam;
    tam = a;
    a = b;
    b = tam;
}
```

Và một hàm main():

```
int main() {
    int x = 1, y = 2;
    swap(x, y);
    printf("x=%d, y=%d", x, y);
    return 0;
}
```

❖ Kết quả?

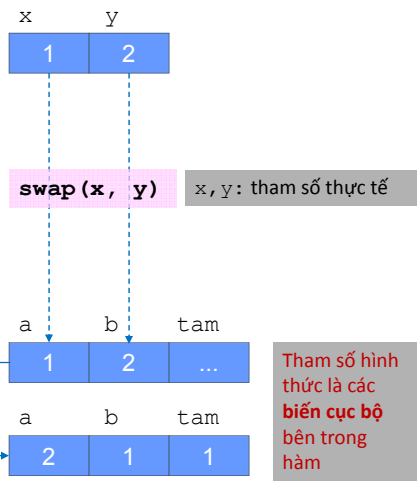


Ví dụ 1 – Giải thích

```
int main() {
    int x = 1, y = 2;
    swap(x, y);
    printf("x=%d, y=%d", x, y);
    return 0;
}
```

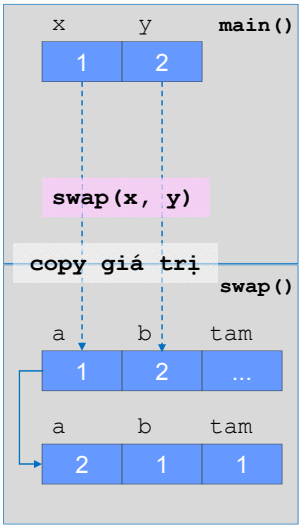
tham số hình thức

```
void swap(int a, int b) {
    int tam;
    tam = a;
    a = b;
    b = tam;
}
```



Tham số hình thức là các **biến cục bộ** bên trong hàm

Ví dụ 1 – Giải thích



1. Tham số hình thức là 1 biến.
2. Giá trị của *tham số thực tế* được truyền (sao chép) cho *tham số hình thức*.
3. Các tác động lên tham số hình thức không ảnh hưởng đến tham số thực tế.

Làm thế nào để thay đổi giá trị của tham số thực tế?

Ví dụ 1' – Cải tiến

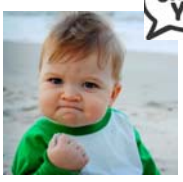
• Viết hàm `swap()` và hàm `main()` lại như sau:

```
void swap(int [ ], int [ ]) {
    int tam;
    tam = [ ];
    [ ] = [ ];
    [ ] = tam;
}
```

```
int main() {
    int x = 1, y = 2;
    swap([      ]);
    printf("x=%d, y=%d", x, y);
    return 0;
}
```

❖Kết quả:

x=2, y=1



Ví dụ 1' – Giải thích

```
int main() {
  int x = 1, y = 2;
  swap(&x, &y);
  printf("x=%d, y=%d", x, y);
  return 0;
}
```

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

- Ký hiệu * được gọi là con trỏ:

  `int *a; //tạo con trỏ a`

- Toán tử & dùng để lấy địa chỉ của biến:

  `&x`: trả về địa chỉ biến `x`

- Truyền địa chỉ một biến cho một con trỏ ⇒ con trỏ trỏ tới biến.

- Phép toán * trên một con trỏ: truy xuất biến mà con trỏ đang **trỏ tới**.

  `tam = *a;` ⇔ `tam = x (=1)`

  `*a = *b;` ⇔ `x = y (=2)`

  `*b = tam;` ⇔ `y = tam (=1)`

2. Khai báo và truy xuất con trỏ

Con trỏ là gì?

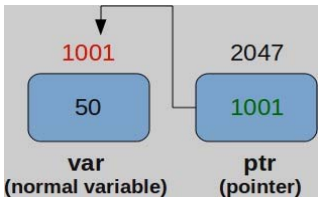
- Con trỏ là 1 biến đặc biệt: chứa địa chỉ của một ô nhớ (hay 1 biến khác)

  vs. biến thường: chứa dữ liệu.

- Khi con trỏ chứa địa chỉ của 1 ô nhớ (biến), *ta nói là*: con trỏ đang trỏ tới (hay tham chiếu tới) ô nhớ (biến) đó.

- **Tính chất**: con trỏ trỏ tới một ô nhớ (biến)
  ⇒ có thể truy xuất ô nhớ (biến) thông qua con trỏ

Con trỏ là gì?



- Ví dụ:
  - Biến `var` có giá trị `50`, được cấp phát vùng nhớ bắt đầu tại địa chỉ `1001`.
  - Biến con trỏ `ptr` được cấp phát vùng nhớ bắt đầu tại địa chỉ `2047` và có giá trị `1001`:
    - Con trỏ `ptr` đang trỏ tới biến `var`.
    - Ta có thể truy xuất biến `var` (vùng nhớ `1001`) thông qua con trỏ `ptr`.

Khai báo con trỏ

- Cú pháp: `<kiểu dữ liệu> *<tên con trỏ>;`
  - Kiểu dữ liệu của một con trỏ: xác định *kiểu biến* hay *kiểu của dữ liệu của vùng nhớ* mà con trỏ có thể trỏ tới.
  - Kiểu dữ liệu của con trỏ có thể là bất kỳ kiểu gì.
  - Tên con trỏ: đặt theo qui tắc đặt tên biến.
  - Có thể khai báo nhiều con trỏ trong một câu lệnh.
- Ví dụ:

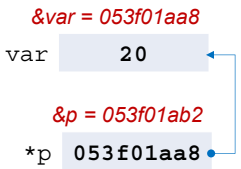
```
int *p;
float *x, *z;
```

Chú ý: Kích thước của các con trỏ luôn bằng nhau, không phụ thuộc vào kiểu dữ liệu của con trỏ.

Các phép toán trên con trỏ

Phép toán	Ý nghĩa
* *(defererence)*	1. Dùng để khai báo một con trỏ. 2. Truy xuất vùng nhớ (biến) con trỏ đang trỏ đến.
& *(reference)*	Lấy địa chỉ của một biến (địa chỉ vùng nhớ được cấp phát cho biến)

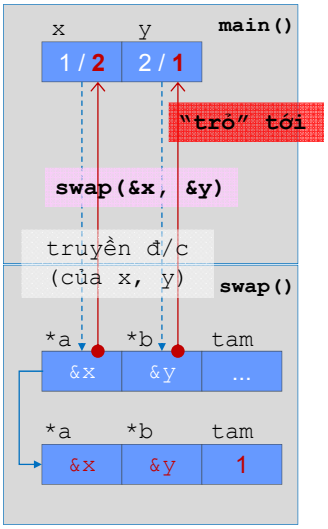
```
int main (){
   int  var = 20;
   int  *p;

   p = &var; //cho p tham chiếu tới var
   printf("%x\n", &var);   ⇒ 53f01aa8
   printf("%x\n", p);      ⇒ 53f01aa8
   printf("%d\n", *p);     ⇒ 20
   printf("%x\n", &p);     ⇒ 53f01ab2
}
```



Các phép toán trên con trỏ (Ví dụ 1')

```
int main() {
    int x = 1, y = 2;
    swap(&x, &y);
    printf("x=%d, y=%d", x, y);
    return 0;
}
```

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



Thực hành 1 (30-45p)

1. Viết chương trình trong đó có biến `short a` với giá trị khởi tạo là 10 và một con trỏ `short *p`.
   - Cho con trỏ `p` trỏ đến `a`.
   - Hiển thị các thông tin:
     - Địa chỉ, giá trị và kích thước của biến `a`
     - Địa chỉ và giá trị của con trỏ `p`
     - Giá trị vùng nhớ mà `p` đang trỏ tới
     - Kích thước của con trỏ `p`
     - Kích thước của dữ liệu mà `p` đang trỏ tới
   - Gán giá trị cho vùng nhớ mà p đang trỏ tới = 10 và hiển thị lại các thông tin trên như trong hình minh họa.

```
C:\Users\TCAN>bt_pointer.exe
&a= 0028FF46
a= 10
sizeof(a)= 2
p= 0028FF46
&p= 0028FF46
*p= 10
sizeof(p)= 4
sizeof(*p)= 2
---------------
&a= 0028FF46
a= 10
p= 0028FF46
&p= 0028FF46
*p= 10
```

Thực hành 1 (30-45p)

2. Viết chương trình đảo hai số như hình bên cạnh:
   - Cho phép người dùng nhập giá trị cho hai biến nguyên x, y
   - Hiển thị giá trị và địa chỉ của các biến x, y
   - Viết hàm đảo giá trị hai số nguyên:
     - Đảo giá trị hai đối số truyền vào cho hàm
     - Bên trong hàm, có hiển thị địa chỉ, nội dung của các đối số
   - Gọi hàm để đảo giá trị của x, y và hiển thị kết quả

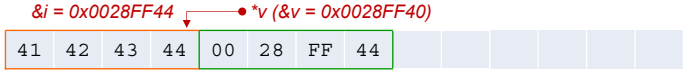
```
C:\Users\TCAN>swap.exe
Nhap x: 5
Nhap y: 10
x=5
y=10
&x=28ff44
&y=28ff40
* Goi ham swap:
a=2686788
b=2686784
&a=28ff20
&b=28ff24
*a=10
*b=5
* Sau khi goi ham swap:
x=10
y=5
&x=28ff44
&y=28ff40
```


Con trỏ void – `void*`

- Là con trỏ chưa có kiểu dữ liệu xác định.
- Có thể cho trỏ tới bất kỳ biến hay vùng nhớ có dữ liệu kiểu gì.
- Khi truy xuất vùng nhớ đang trỏ tới bởi con trỏ, cần phải ép kiểu một cách tường minh.

```
int main (){
   int i = 0x41424344; //hệ 16
   void *v = &i;
   printf("%p, %p", *(short*)v, *(int*)v);
}
```

⇒ 00004344, 41424344



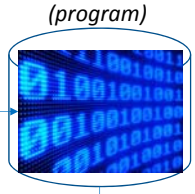
Cấu trúc vùng nhớ của tiến trình

mã nguồn
(program source code)

```
#include <stdio.h>
int main() {
   printf("Hello World");
   return 0;
}
```

biên dịch

chương trình
(program)

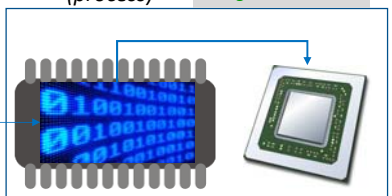


Tập hợp các chỉ thị (instructions) có liên quan, được tổ chức một cách logic để thực hiện một tác vụ nào đó.

thực thi

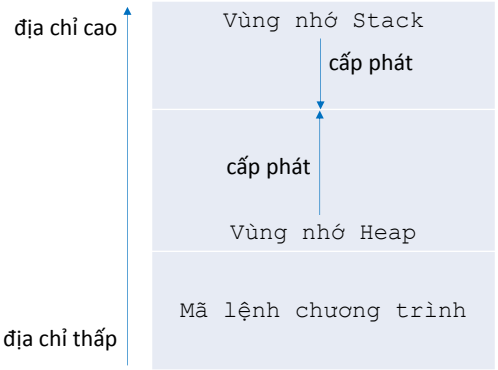
tiến trình
(process)

Một chương trình đang thực thi



Thực thi chương trình:
1. Mã lệnh chương trình (chỉ thị) được tải vào bộ nhớ.
2. CPU sẽ lấy lệnh từ bộ nhớ để thực thi.

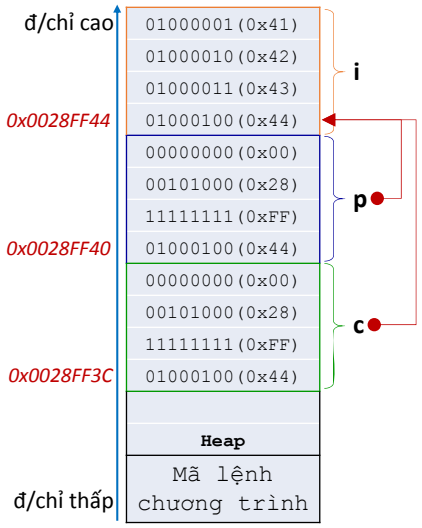
Cấu trúc vùng nhớ của tiến trình



- Vùng nhớ stack:
  - cấp phát cho các biến cục bộ,
  - từ địa chỉ cao xuống thấp.
- Vùng nhớ heap:
  - dùng để cấp phát "động" (giới thiệu sau),
  - từ địa chỉ thấp lên cao.
- Còn một số phân đoạn khác (biến toàn cục,...)
- **Lưu ý**: các vùng nhớ này có thể không liên tục (vùng nhớ heap có thể "nằm trên" vùng nhớ stack).

Cấu trúc vùng nhớ của tiến trình

```
int main (){
   int i = 0x41424344; //hệ 16
   int *p = &i;
   char *c = &i;
   printf("%p, %c", *p, *c);
}
```



Cấp phát vùng nhớ động

- Con trỏ không chỉ có thể trỏ đến một biến mà còn có thể trỏ đến các vùng nhớ được cấp phát "động".
- Cấp phát vùng nhớ động:
  - `void* malloc(size_t size)`
  - `void* calloc(size_t num, size_t size)`
- Thu hồi vùng nhớ được cấp phát động:
  - `void free(void *ptr)`

Chú ý: `size_t` là một bí danh (alias) của kiểu `unsigned int` (hệ thống 32bit) hoặc `unsigned long` (64bit).

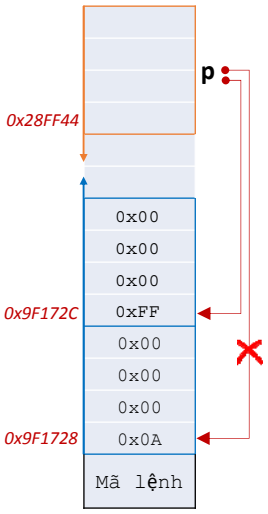
Cấp phát vùng nhớ động

- Chú ý:
  - Vùng nhớ cấp phát động sẽ được cấp phát trong vùng nhớ heap (từ thấp đến cao).
  - Hàm `malloc()` thường được sử dụng kèm với toán tử `sizeof()`, dùng để lấy kích thước của một kiểu dữ liệu hoặc biến.
  - Các vùng nhớ cấp phát động không được tự động giải phóng khi chương trình kết thúc
    ⇒ phải giải phóng tường minh bằng hàm `free()` để tránh rò rỉ bộ nhớ.
  - **Ứng dụng**: tạo các "mảng động", chuỗi có kích thước "động",...

Cấp phát vùng nhớ động

```
int main (){
    int *p = (int*)malloc(sizeof(int));
    *p = 10;
    printf("%p, %p, %d\n", p, &p, *p);
    //kết quả:                ???

    p = (int*)malloc(sizeof(int));
    *p = 15;
    printf("%p, %p, %d\n", p, &p, *p);
    //kết quả:                ???
}
```

Chương trình trên vi phạm nguyên tắc gì trong việc sử dụng vùng nhớ heap (cấp phát động)? Cách giải quyết?



3. Ứng dụng của con trỏ

Ứng dụng của con trỏ

1. Cấp phát động bộ nhớ:
   - Mảng động: mảng có kích thước có thể thay đổi
2. Chuỗi ký tự "động" (phần sau)
3. Truyền tham số cho hàm: cho phép thay đổi giá trị của tham số thực tế
4. Con trỏ trỏ đến cấu trúc
5. Trị trả về của hàm là con trỏ (phụ lục)

Tại sao cần "mảng động"?

- Mảng: tập hợp có thứ tự các phần tử có cùng kiểu dữ liệu.
- **Tình huống**: Viết chương trình quản lý sinh viên (tạo, thêm, sửa, xóa,...)
  - Cần 1 mảng để lưu trữ danh sách sinh viên.
  - Tạo mảng bao nhiêu phần tử? 10, 100, 200, 500,...?
    - Kích thước lớn: không sử dụng hết sẽ hao phí bộ nhớ.
    - Kích thước nhỏ: có trường hợp không đủ không gian lưu trữ.
- **Giải pháp**: Tạo mảng sau khi có số phần tử!?
  - Nếu có thêm/xóa sinh viên sẽ giải quyết như thế nào?

Khái niệm "mảng động"

- Mảng động:
  - mảng chưa có kích thước xác định khi khai báo.
  - sẽ được cấp phát không gian lưu trữ khi kích thước đã xác định.
  - có thể cấp phát lại vùng nhớ khi cần thay đổi số lượng phần tử của mảng.
- **Chú ý**: Việc cấp phát vùng nhớ mới đòi hỏi
  - việc copy dữ liệu từ vùng nhớ cũ sang vùng nhớ mới
  - giải phóng vùng nhớ củ nếu không còn sử dụng.

Tạo mảng "động" với con trỏ

- Tạo mảng động:
  1. Khai báo một con trỏ (sẽ được sử dụng như là mảng).

     `<kiểu phần tử mảng> *<tên mảng>;`

  2. Khi biết được số phần tử của mảng, cấp phát 1 vùng nhớ có kích thước tương ứng.

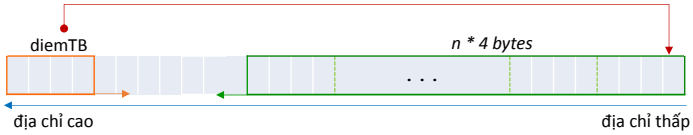
     `<tên mảng> =(<kiểu phần tử>*)malloc(kích thước);`

     `<tên mảng> =(<kiểu phần tử>*)`
     `calloc(số phần tử, kích thước 1 phần tử);`

- **Chú ý**: dùng toán tử **`sizeof`** để tính kích thước một phần tử của mảng.

Ví dụ

```
int main() {
    float *diemTB;
    int n = 0;
    printf("Nhap so sinh vien: ");
    scanf("%d", &n);    //giả sử nhập 4
    diemTB = (float*)malloc(sizeof(float) * n);
    //...

    free(diemTB);
}
```



Truy xuất các phần tử của mảng

- Dùng chỉ số (giống như mảng "thường"):

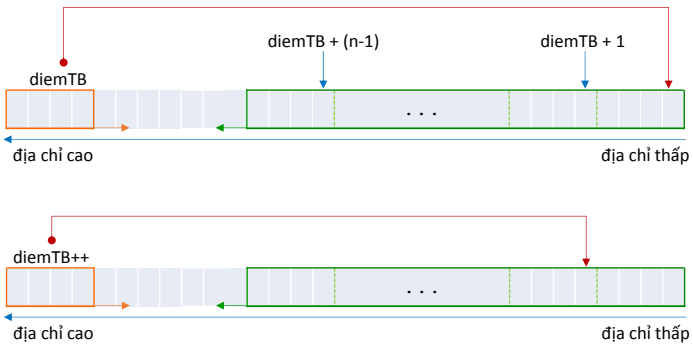
```
diemTB[0] = 8.5;
diemTB[1] = 9.0;
```

- Dùng phép toán dereference (*):

```
*diemTB = 8.5;
*(diemTB + 1) = 9.0;
```


"Dời" con trỏ (pointer movement)

- Phép toán: +, –, ++, ––
- Đơn vị độ dời: bằng kích thước 1 phần tử mà con trỏ đang trỏ đến.



Duyệt mảng (array iteration)

- Có 3 cách:
  - Dùng chỉ số

```
for (i=0, i<n; i++) {
   diemTB[i] = ...;
   //...
}
```

  - Dùng toán tử dời con trỏ:

```
float *q = p;
for (i=0; i<n; i++, q++) {
   *q = ...;
   //...
}
```

```
for (i=0; i<n; i++) {
   *(diemTB + i) = ...;
   //...
}
```


So sánh mảng và con trỏ

Đặc điểm	Mảng thường	Mảng con trỏ
Khác nhau		
Không gian lưu trữ	Phải khai báo lúc tạo mảng	Có thể cấp phát sau
Vị trí lưu trữ	Vùng nhớ Stack ⇒ Không gian lưu trữ được tự động thu hồi khi ra khỏi phạm vi khai báo mảng	Vùng nhớ Heap ⇒ Không được tự động thu hồi khi ra khỏi phạm vi khai báo, và cả khi kết thúc chương trình
Không gian lưu trữ	Không thay đổi được	Có thể thay đổi được
Giống nhau		
Mục đích sử dụng	Dùng để lưu trữ 1 tập có thứ tự các phần tử cùng kiểu.	
Cấu trúc dữ liệu	Về bản chất đều là con trỏ. Tuy nhiên, mảng thường là một con trỏ chỉ trỏ đến 1 vùng nhớ duy nhất (hằng con trỏ).	
Tên	Tên mảng được sử dụng như tên một con trỏ `char s[20]; scanf("%s", s);  //not &s`	

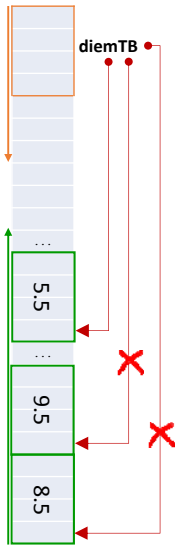
Mảng động & Cấp phát động bộ nhớ

- Mảng động mà một trường hợp của cấp phát động bộ nhớ.
- Tạo sao cần cấp phát động bộ nhớ?
  - Mảng động là 1 trường hợp của cấp phát động bộ nhớ:
    - Cấp phát theo yêu cầu
    - Có thể "co dãn" vùng nhớ lưu trữ dữ liệu.
    - Yêu cầu bộ nhớ phải liên tục nhau.
  - Cấp phát động không chỉ cấp phát một mảng các đối tượng mà có thể cấp phát từng phần tử:
    - Tối ưu hóa việc sử dụng bộ nhớ, tránh dư thừa.
    - Khắc phục tình trạng "phân mảnh" bộ nhớ.

Mảng động & Cấp phát động bộ nhớ

```
int main () {
   float *diemTB;
   int count = 0;
   do {
     float tam;
     printf("Nhap diem cho SV moi (-1: thoat): ");
     scanf("%f", &tam);
     if (-1 == tam)
        break;
     count++;
     diemTB = (float*)malloc(sizeof(float));
     *diemTB = tam;
   } while (true);
   printf("Tong so SV: %d", count);
}
```

```
Nhap diem cho SV moi (-1: thoat): 8.5
Nhap diem cho SV moi (-1: thoat): 9.5
Nhap diem cho SV moi (-1: thoat): 5.5
Nhap diem cho SV moi (-1: thoat): -1
Tong so SV: 3
```



Làm thế nào để truy xuất vùng nhớ bị "bỏ rơi"?

Thực hành 2 (60p)

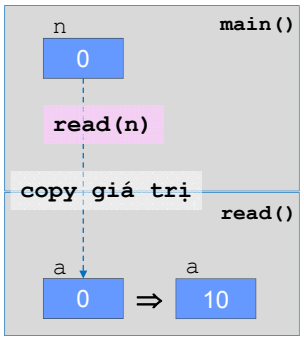
1. Làm bài tập số 3 trong phần Bài tập tổng kết
2. Viết một chương trình quản lý điểm trung bình của sinh viên được mô tả như sau:
   1. Cho phép người dùng nhập vào số lượng sinh viên.
   2. Tạo một mảng `diemTB` kiểu `float` để lưu trữ điểm trung bình.
   3. Cho phép người sử dụng nhập điểm TB cho các SV.
   4. Hiển thị số sinh viên có điểm trung bình trên 5.0.
   5. Cho phép người dùng nhập số lượng sinh viên cần thêm vào.
   6. Chỉnh sửa lại mảng `diemTB` để có thể nhận thêm điểm TB của các sinh viên mới.
   7. Nhập điểm trung bình cho các SV mới và thực hiện lại bước 5.

Tại sao cần truyền t/số bằng con trỏ?

- Bài toán: Viết 1 hàm nhận vào 1 biến kiểu `int`, yêu cầu người dùng nhập vào giá trị cho biến đó.

```
#include <stdio.h>
void readInt(int a) {
   printf("Nhap 1 so nguyen: ");
   scanf("%d", &a);
}

int main() {
   int n = 0;
   readInt(n);
   printf("n=%d", n);
}
```

```
Nhap vao 1 so nguyen: 10
n=0
```



Giải pháp

- Biến `a` (tham số hình thức) phải là một con trỏ, "trỏ" đến biến `n` (tham số hình thức):
  1. Tham số hình thức `a` phải được khai báo là một con trỏ (`int *a`).
  2. Trong hàm `main()`, phải truyền địa chỉ của biến `n` (tham số thực tế) cho tham số hình thức `a`.
  3. Trong hàm `readInt()`, thay đổi giá trị của `n` thông qua con trỏ `*a`.
- Cách truyền tham số này được gọi là "truyền bằng con trỏ" hay "truyền bằng địa chỉ".

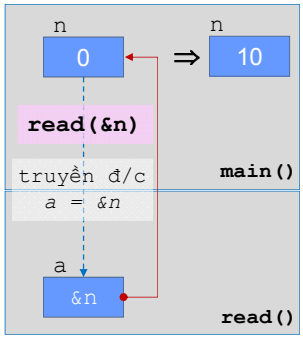
Giải pháp

truyền bằng con trỏ

```
#include <stdio.h>
void readInt(int *a) {
   printf("Nhap 1 so nguyen: ");
   scanf("%d", a);
}
int main() {
   int n = 0;
   readInt(&n);
   printf("n=%d", n);
}
```

truyền địa chỉ `n` cho `a`

```
Nhap vao 1 so nguyen: 10
n=10
```



Hãy kể tên một số hàm đã học có sử dụng cách truyền tham số bằng con trỏ?

Con trỏ đến cấu trúc

- Một con trỏ có thể trỏ đến một biến/vùng nhớ có dữ liệu thuộc bất kỳ kiểu gì

  ⇒ một con trỏ có thể trỏ đến một cấu trúc hoặc sử dụng như là một mảng động các phần tử kiểu cấu trúc.

- Ví dụ:

```
typedef struct {
  char hoten[30];
  char ngaysinh[11];
  float diemTBTL;
} Sinhvien;

Sinhvien sv1;
Sinhvien *p = &sv1;
```

```
dssv = (Sinhvien*)malloc(
            sizeof(Sinhvien)*10);
//hoặc
dssv = (Sinhvien*)calloc(10,
              sizeof(Sinhvien));
```


Truy xuất các phần tử của cấu trúc

- Có 2 cách:
  - Dùng toán tử * (dereference) kết hợp với toán tử . (dot)

```
(*p).diemTBTL = 8.5;
strcpy((*p).hoten, "TCAN");
strcpy((*p).ngaysinh, "23/12/78");
```

  - Dùng toán tử -> (arrow operator)

```
p->diemTBTL = 8.5
strcpy(p->hoten, "TCAN");
strcpy(p->ngaysinh, "23/12/78");
```

Truy xuất các phần tử của cấu trúc

- Lưu ý: khi dùng con trỏ như là mảng, một phần tử của mảng (truy xuất bằng toán tử `[]`) là một cấu trúc chứ không phải là một con trỏ.

```
for (i=0; i<10; i++) {
  dssv[i].diemTBTL = 0;
  strcpy(dssv[i].hoten, "");
  //...
}
```

Phần tử thứ i của mảng, là một cấu trúc

sai

```
dssv[i]->diemTBTL = 0;
strcpy(dssv[i]->hoten, "");
```

- Hoặc:

```
for (i=0; i<10; i++) {
  (dssv+i)->diemTBTL = 0;
  strcpy((dssv+i)->hoten, "");
  //...
}
```

một con trỏ, trỏ tới phần tử thứ i của mảng

Con trỏ nâng cao

1. Con trỏ hàm (function pointer):
   - Một con trỏ có thể trỏ đến một hàm.
   - Ta có thể gọi hàm thông qua con trỏ.
   - Thường được sử dụng trong hàm callback.

2. Con trỏ của con trỏ:
   - Một con trỏ có thể trỏ đến con trỏ khác.

     `int **p;` //`p` là con trỏ có thể trỏ đến các con trỏ `int *p`
   - Thường được sử dụng để tạo mảng động nhiều chiều, truyền đối số cho hàm bằng con trỏ với đối số là một con trỏ,...

Các lỗi thường gặp khi s/dụng con trỏ

1. Sử dụng con trỏ chưa khởi tạo:

```
int main() {
   int *p;  // khai báo con trỏ nhưng ko khởi tạo
   *p = 42; //điều gì xảy ra nếu p đang trỏ vào vùng
            //nhớ chứa mã lệnh của chương trình?
   //...
}
```

2. Không giải phóng vùng nhớ cấp phát động:

```
int main() {
   int* p = (int*)malloc(sizeof(int) * 1000);
   //sử dụng con trỏ...
   return 0; //không gọi hàm free() cho p
}
```


Tổng kết

- Con trỏ là 1 biến đặc biệt, dùng để chứa địa chỉ của một biến hay một vùng nhớ.
- Khai báo con trỏ: `<datatype> *<pointer name>;`
- Hai phép toán cơ bản trên con trỏ là `*` và `&`
- Con trỏ chứa đ/chỉ vùng nhớ (biến) nào, ta nói con trỏ đang trỏ (tham chiếu) tới vùng nhớ (biến) đó.
- Ta có thể truy xuất một vùng nhớ thông qua một con trỏ đang trỏ tới vùng nhớ đó.
- Con trỏ `void*` có thể trỏ đến bất kỳ biến kiểu gì.

Tổng kết

- Con trỏ có thể được sử dụng như mảng động bằng cách cấp phát vùng nhớ động.
- Vùng nhớ cấp phát động nằm trong vùng nhớ heap, sẽ không được giải phóng tự động.
- Mảng (thường) là một hằng con trỏ.
- Truyền tham số cho hàm bằng con trỏ cho phép hàm thay đổi giá trị tham số thực tế.
- Con trỏ phải được khởi tạo trước khi sử dụng.
- Vùng nhớ cấp phát động phải được giải phóng một cách tường minh dùng hàm `free()`.

Check what you achieved.
Re-learn what you haven't mastered.

4. Kiểm tra kiến thức & Bài tập

Kiểm tra kiến thức

1. Khai báo con trỏ nào sau đây là đúng?
   a) `int x;`
   b) `int &x;`
   c) `ptr x;`
   d) `int *x;`
2. Lệnh nào sau đây dùng để lấy địa chỉ của biến `a`?
   a) `*a;`
   b) `a;`
   c) `&a;`
   d) `address(a);`

Kiểm tra kiến thức

3. Lệnh nào sau đây trả về địa chỉ của biến đang được trỏ tới bởi con trỏ `a`?
   a) `a;`
   b) `*a;`
   c) `&a;`
   d) `address(a);`
4. Lệnh nào sau đây trả về giá trị của biến mà con trỏ `a` đang trỏ tới?
   a) `a;`
   b) `val(a);`
   c) `*a;`
   d) `&a;`

Kiểm tra kiến thức

5. Hàm/Lệnh nào dưới đây dùng để cấp phát vùng nhớ động trong C?
   a) new
   b) malloc
   c) create
   d) value
6. Lệnh nào sau đây dùng để giải phóng một vùng nhớ được cấp phát động?
   a) free
   b) delete
   c) clear
   d) remove

Kiểm tra kiến thức

7. Hàm và con trỏ trong C có thể được sử dụng thay thế lẫn nhau?
   a) Đúng
   b) Sai
8. Cho p là một con trỏ kiểu `int` và `i` là một biến kiểu `int`. Câu lệnh nào sau đây là đúng?
   a) `p = 0;`
   b) `p = i;`
   c) `p = p + 1;`
   d) `p = &i;`

Kiểm tra kiến thức

9. Lệnh `int *x;` và `int* x;` cho kết quả giống nhau?
   a) Đúng
   b) Sai
10. Rò rỉ bộ nhớ xảy ra khi ta cấp phát vùng nhớ động bằng hàm malloc() và sau đó giải phóng bằng hàm `free()`.
    a) Đúng
    b) Sai
11. Con trỏ có thể trỏ đến kiểu dữ liệu nào sau đây?
    a) int
    b) char
    c) structs
    d) Tất cả các kiểu trên

Kiểm tra kiến thức

12. Now that the pointer is declared, let's set it to point to something: `int a = 5;`
    a) `*fun_int = &a;`
    b) `*fun_int = *a;`
    c) `fun_int = *a;`
    d) `fun_int = &a;`
13. Now let's dereference the pointer:
    a) `*fun_int = 12;`
    b) `.fun_int = 12;`
    c) `&fun_int = 12;`

Kiểm tra kiến thức

14. Hãy cho biết kết quả thực thi các chương trình sau:

```
#include <stdio.h>
void fun(int *ptr) {
    *ptr = 30;
}

int main() {
   int y = 20;
   fun(&y);
   printf("%d", y);

   return 0;
}
```

a. 20
b. 30
c. Compiler error
d. Runtime error

```
#include <stdio.h>
void fun(int x) {
    x = 30;
}

int main() {
   int y = 20;
   fun(y);
   printf("%d", y);

   return 0;
}
```

a.20
b.30
c.Compiler error
d.Runtime error

Kiểm tra kiến thức

15. Hãy cho biết kết quả thực thi của chương trình sau:

```c
#include <stdio.h>
int main() {
   int *ptr;
   int x;

   ptr = &x;
   *ptr = 0;
   printf(" x = %d\n", x);
   printf(" *ptr = %d\n", *ptr);

   *ptr += 5;
   printf(" x  = %d\n", x);
   printf(" *ptr = %d\n", *ptr);

   (*ptr)++;
   printf(" x = %d\n", x);
   printf(" *ptr = %d\n", *ptr);
   return 0;
}
```

a)

```
x = 0
*ptr = 0
x = 0
*ptr = 0
x = 0
*ptr = 0
```

b)

```
x = garbage value
*ptr = 0
x = garbage value
*ptr = 5
x = garbage value
*ptr = 6
```

c)

```
x = 0
*ptr = 0
x = 5
*ptr = 5
x = 6
*ptr = 6
```

d)

```
x = 0
*ptr = 0
x = 5
*ptr = 5
x = garbage value
*ptr = garbage value
```


Bài tập tổng kết

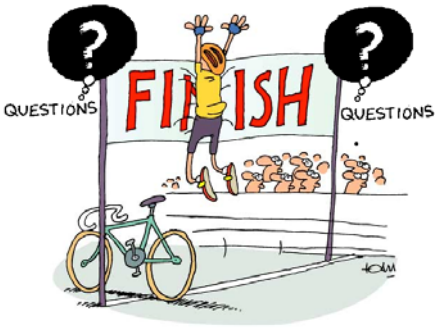
1. Viết lại hàm `swap()` theo các cách truyền tham số đã học và một chương trình chính để sử dụng các hàm vừa viết để nhận xét về các cách truyền tham số.
2. Viết 1 hàm nhận vào 3 đối số `a, b, c` và bên trong hàm sẽ xoay vòng giá trị 3 đối số này: `a=b, b=c, c=a`. Viết một chương trình chính để kiểm tra sự hoạt động của hàm.
3. Viết một hàm nhận vào một con trỏ `int*`, trỏ tới một mảng các số nguyên, và kích thước của mảng. Hàm trả về số số chẵn trong mảng.
   Chú ý: chỉ sử dụng con trỏ và các toán tử trên con trỏ, không sử dụng toán tử `[]` của mảng.

Bài tập tổng kết

4. Viết chương trình theo yêu cầu như sau:
   - Viết hàm `sort()` nhận vào 3 đối số kiểu `int a, b, c`. Hàm này gán giá trị lớn nhất trong 3 số này cho `a`, giá trị lớn kế tiếp cho `b` và giá trị nhỏ nhất cho `c`.
   - Chương trình chính:
     - Tạo mảng gồm 3 phần tử kiểu int và nhập giá trị cho mảng.
     - Truyền các phần tử này vào cho hàm sort().
     - Hiển trị giá trị của mảng sau khi gọi hàm.
5. Làm tiếp b/tập số 2 của bài Thực hành 2 theo y/cầu sau:
   - Viết một hàm `trenTB(...)` nhận vào một mảng và số phần tử của mảng. Hàm trả về số lượng SV trên trung bình (>= 5.0).
   - Trong yêu cầu số 4, gọi hàm `trenTB()` để hiển thị số sinh viên có điểm trung bình trên 5.0.

Bài tập tổng kết

6. Viết một chương trình quản lý sinh viên như sau:
   1. Tạo 1 cấu trúc **`Sinhvien`** có thể lưu trữ thông tin của một SV bao gồm: MSSV, họ tên, lớp, số TC tích lũy, điểm TB tích lũy.
   2. Cho phép người dùng nhập vào số lượng SV ***n***
   3. Tạo một mảng động gồm `n` phần tử kiểu `Sinhvien`
   4. Cho phép người dùng nhập thông tin cho các SV.
   5. Hiển thị SV có số điểm cao nhất và thấp nhất **(*)**.
   6. Cho phép người dùng thêm vào **`m`** SV với `m` nhập từ bàn phím (*Hướng dẫn: cấp phát mảng mới + copy dữ liệu + xóa mảng cũ + nhập thông tin cho các SV mới*).
   7. Hiển thị thông tin của tất cả các SV ra màn hình cùng với xếp loại của SV.



CT101 – Lập trình căn bản

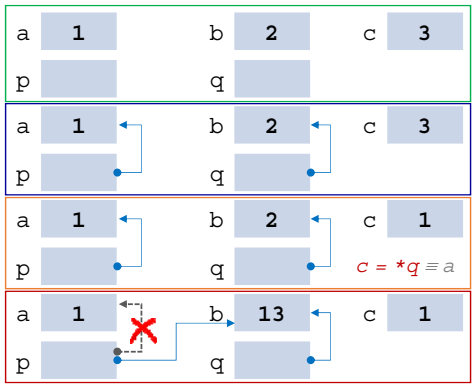
Khoa CNTT&TT – ĐHCT

5. Phụ lục

Phép gán con trỏ

- Gán hai con trỏ: cho hai con trỏ trỏ cùng một nơi.

```
#include <stdio.h>
int main() {
    int a=1, b=2, c=3;
    int *p, *q;
    p = &a;
    q = &b;
    c = *p;

    p = q;
    *p = 13;
}
```

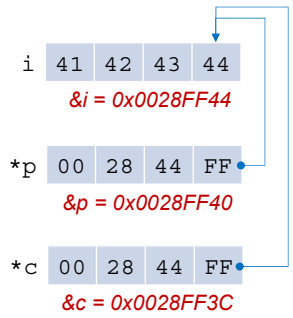


Kiểu của con trỏ

- Kích thước tất cả các con trỏ đều bằng nhau
  - ⇒ Tại sao cần kiểu của con trỏ?
- Kiểu của con trỏ qui định cách thức truy xuất vùng nhớ (biến) con trỏ đang trỏ tới.

```
int main (){
    int i = 0x41424344; //hệ 16
    int *p = &i;
    char *c = &i;
    printf("%p, %c", *p, *c);
}
```

- Kết quả? [redacted]



Hằng con trỏ và con trỏ (đến) hằng

- Constant pointer (hằng con trỏ): không thể thay đổi vị trí vùng nhớ tham chiếu.

```
<kiểu dữ liệu>* const <tên con trỏ>
```

  - Chú ý: phải khai báo vị trí tham chiếu của con trỏ trong câu lệnh khai báo.

- Pointer to constant (con trỏ hằng): không thể thay đổi giá trị vùng nhớ con trỏ đang tham chiếu.

```
const <kiểu dữ liệu> *<tên con trỏ>;
```

  - Chú ý: Có thể thay đổi vị trí tham chiếu của con trỏ hằng

Hằng con trỏ và con trỏ (đến) hằng

Hằng con trỏ

```
int main() {
   int var1=1, var2=2;
   int *const ptr1;   //lỗi
   int *const ptr2 = &var1;

   *ptr2 = 10;        //OK
   ptr2 = &var2;      //lỗi
   printf("%d\n", *ptr2);
}
```

Con trỏ hằng

```
int main() {
   int var1=1, var2=2;
   const int* ptr;

   ptr = &var1;
   *ptr = 10;        //lỗi
   ptr = &var2;      //OK
   printf("%d\n", *ptr);
}
```

- Có thể kết hợp con trỏ hằng và hằng con trỏ:

```
const <kiểu dữ liệu>* const <tên con trỏ>;
```