C Intermediate

1. Typedef

Là lệnh dùng để đặt tên mới cho kiểu dữ liệu có sẵn.

Cú pháp:

```
typedef kiểu_cũ kiểu_mới;
typedef unsigned char uint8_t;
```

2. Struct

Là một kiểu dữ liệu phức hợp được tạo ra từ các kiểu dữ liệu khác.

Cú pháp định nghĩa:

```
struct tên_struct {
   Khai báo các biến thành phần
};
```

Vd:

```
struct rectangle {
   float heigh;
   float width;
};
```

Thông thường kết hợp dùng typedef để định nghĩa struct nhằm dễ khai báo

```
typedef struct {
   float height;
   float width;
} rectangle;
```

Khai báo biến (sau khi dùng với typedef):

```
rectangle ABCD;
rectangle *MNPQ;
```

Để truy xuất các thành phần của struct, người ta dùng dấu chấm (.):

```
ABCD.height = 3;
ABCD.width = 9;
```

Với trường hợp đối tượng là con trỏ, dùng dấu mũi tên (->) để truy xuất thành phần:

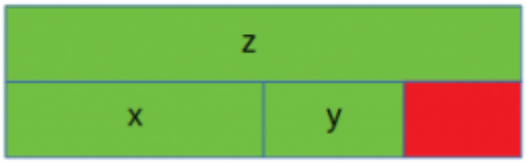
```
MNPQ->height = 3;
MNPQ->width = 9;
```

Chú ý: kích thướng của struct thường lớn hơn bằng tổng kích thước các thành phần của nó do cơ chế thêm padding nhằm tránh những vấn đề về data alignment

```
struct A {
    // sizeof(int) = 4
    int x;
    // Padding of 4 bytes

    // sizeof(double) = 8
    double z;

    // sizeof(short int) = 2
    short int y;
    // Padding of 6 bytes
};
```

Với cách khai báo như trên, do kích thước của z lớn hơn x nên sẽ có padding (màu đỏ) sau biến x và y, độ lớn của struct là 24 bytes.

```
struct B {
    // sizeof(double) = 8
    double z;

    // sizeof(int) = 4
    int x;

    // sizeof(short int) = 2
    short int y;
    // Padding of 2 bytes
};
```



Bằng cách đổi thứ tự khai báo biến theo thứ tự độ lớn giảm dần, ta sẽ giảm được kích thước bộ nhớ cần cho struct thành 16 bytes.

3. Enum

Là kiểu dữ liệu liệt kê. Dùng để tập hợp các hằng số có ý nghĩa tương đương nhau thành những nhóm hằng số riêng biệt.

Cú pháp định nghĩa:

```
enum tên_enum {
   Danh sách các giá trị
};
```

Vd:

```
enum weekday {
   mon, tue, wed, thu, fri, sat, sun
};
```

Không cần chúng ta trực tiếp gán giá trị cho các tên hằng số, **compiler** đã tự động khởi tạo giá trị cho chúng, bắt đầu với giá trị 0 và tăng dần, nghĩa là mon = 0, tue = 1, wed = 2,…

Tương tự struct người ta cũng kết hợp typedef với enum:

```
typedef enum {
   mon, tue, wed, thu, fri, sat, sun
} weekday;
```

Khai báo biến:

```
weekday today = sun;
```

4. Pointer

Mỗi biến khi được khai báo đều được cấp phát cho 1 vùng nhớ nhất định ở những nơi (địa chỉ) khác nhau. Biến con trỏ là biến dùng để lưu trữ địa chỉ của các biến đó.

Cú pháp khai báo:

```
int a;     // a là số nguyên 32-bit
int *p;    // p là con trỏ lưu địa chỉ của biến int
```

(*) : toán tử lấy giá trị tại địa chỉ lưu trong con trỏ.

(&) : toán tử lấy địa chỉ của biến

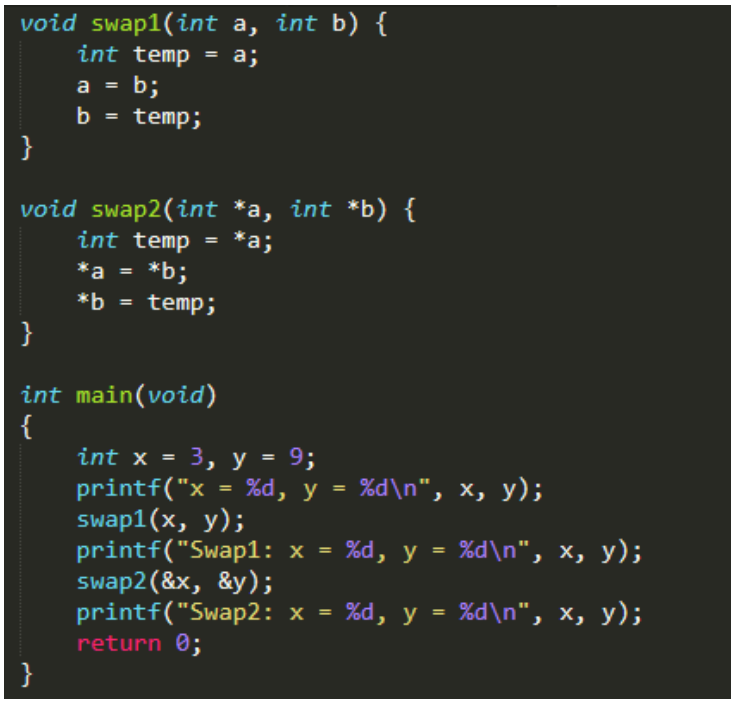
Vd :

```
p = &a ;    // Lấy địa chỉ của a lưu vào con trỏ p
```

```
    int b = *p;      // Lấy giá trị của ô nhớ có địa chỉ lưu trong p gán vào b
```

Tùy theo hệ điều hành mỗi biến con trỏ, dù là con trỏ thuộc kiểu nào (int, float, double,…) cũng chỉ chiếm số lượng byte giống nhau. Hệ điều hành 32 bit thì biến con trỏ chiếm 4 byte, hệ điều hành 64 bit thì biến con trỏ chiếm 8 byte.

5. Tham chiếu, tham trị



```c
void swap1(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

void swap2(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main(void)
{
    int x = 3, y = 9;
    printf("x = %d, y = %d\n", x, y);
    swap1(x, y);
    printf("Swap1: x = %d, y = %d\n", x, y);
    swap2(&x, &y);
    printf("Swap2: x = %d, y = %d\n", x, y);
    return 0;
}
```

Ở hàm swap1 ta truyền vào giá trị (tham trị) của x và y chứ không phải 2 biến x, y nên giá trị của chúng không thay đổi khi hàm kết thúc.

Hàm swap2 ta truyền vào địa chỉ (tham chiếu) của 2 biến x và y và thực hiện đổi chỗ 2 giá trị lưu tại 2 địa chỉ đó thông qua con trỏ nên sau khi hàm kết thúc 2 biến x và y đổi giá trị cho nhau. Do tham số của hàm là con trỏ (a và b) nên khi gọi hàm ta truyền vào địa chỉ của 2 biến x và y.

➔ Khi nào cần truyền tham chiếu?

- Khi cần thay đổi giá trị của biến được truyền vào hàm

- Khi làm việc với các đối tượng kích thước lớn. Truyền tham chiếu tăng tốc độ thực thi chương trình
- Khi muốn "trả về" nhiều giá trị

6. Con trỏ và mảng

Xét chương trình sau:

```
int main(){
    // Khai báo mảng có 4 phần tử
    int A[] = {5, 8, 2, 1};

    printf("\nDia chi cua mang A = %X\n", &A);
    printf("Gia chi cua mang A = %X\n", A);
    // Thử in địa chỉ của từng phần tử
    // sizeof (A): Kích thước của mảng
    // sizeof (int): Kích thước của kiểu int
    for(int i = 0; i < sizeof (A) / sizeof (int); i++){
        printf("Dia chi cua A[%d] = %X\n", i, &A[i]);
    }
}
```

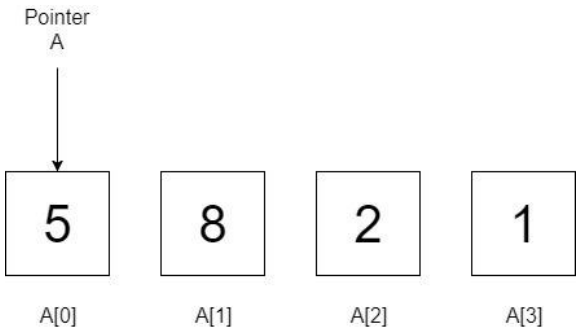
Kết quả:

```
Dia chi cua mang A = DEF08700
Gia chi cua mang A = DEF08700
Dia chi cua A[0] = DEF08700
Dia chi cua A[1] = DEF08704
Dia chi cua A[2] = DEF08708
Dia chi cua A[3] = DEF0870C
```

Kết quả cho thấy địa chỉ của các phần tử trong mảng cách nhau 4 giá trị tương ứng với 4 bytes (kích thước của biến int). Từ đó suy ra các phần tử trong mảng xếp cạnh nhau trong bộ nhớ.

Và giá trị của arr đúng bằng địa chỉ mảng và bằng địa chỉ của phần tử đầu tiên trong mảng: `&A[0]` tương đương `&A` và tương đương `A`.

Có thể sử dụng các toán tử +, -, ++, -- khi thao tác với con trỏ.



`A+1` đưa con trỏ sang ô nhớ liền kề chứ không tăng địa chỉ của nó lên 1. Trong trường hợp con trỏ kiểu int, giá trị của nó tăng lên 4.

- `&A[0]` và `A` có cùng giá trị, và `A[0]` hay `*A` là tương đương nhau.
- `&A[1]` tương đương với `A+1` và `A[1]` tương đương với `*(A+1)`.
- `&A[2]` tương đương với `A+2` và `A[2]` tương đương với `*(A+2)`.

7. Tiền xử lý

Tất cả các lệnh tiền xử lý bắt đầu bằng một dấu thăng (#) và không kết thúc bằng dấu chấm phẩy (;)

Các lệnh tiền xử lý thường phân thành 3 nhóm chính:

- Chèn tệp (#include)
- Định nghĩa (#define)
- Chỉ thị biên dịch theo điều kiện (#if, #else, #endif,…)

#include cho phép chèn nội dung một file khác vào file đang viết (chèn thư viện).

```
#include <header.h>
#include “header.h”
```

Cách trên dùng để thêm các file có trong include directory hoặc của complier, thường là những thư viện có sẵn như stdio, math, stdlib, string,…

Cách dưới dùng để thêm các file trong project của người dùng, xác định bằng đường dẫn tương đối so với fie hiện tại.

File người dùng tạo ra vẫn có thể dùng `#include <>` nếu thư mục chứa file đó được thêm vào include directory.

#define cho phép định nghĩa một số, một chuỗi, một biểu thức bằng một tên nào đó. Trong chương trình, bất kỳ chỗ nào muốn sử dụng chuỗi hay biểu thức này, ta chỉ cần ghi tên macro đặc trưng của nó. Trước khi dịch chương trình, bộ tiền xử lý sẽ thay thế các macro bằng biểu thức thật của nó.

```
#define identifier
#define identifier substitution_text
```

VD:

```
#define MAX 10
```

Khi đó, nếu gặp MAX ở bất kỳ vị trí nào trong chương trình, C sẽ thay bằng 100.

Chú ý khi tạo thư viện:

Nếu một tập tin muốn thực hiện include hai lần, trình biên dịch sẽ xử lý nội dung của nó hai lần và nó sẽ dẫn đến lỗi. Cách để ngăn chặn điều này là kẹp toàn bộ nội dung thực của tệp trong một điều kiện, như thế này:

```
#ifndef HEADER_FILE
#define HEADER_FILE
      The header file
#endif
```

Khi file được include một lần nữa, điều kiện sẽ là sai, bởi vì HEADER_FILE được định nghĩa. Bộ tiền xử lý sẽ bỏ qua toàn bộ nội dung của tệp và trình biên dịch sẽ không nhìn thấy nó hai lần.

8. Bài tập

Bài 1 : Viết chương trình in ra hình tam giác có chiều cao cho trước

- Input: chiều cao n=4
- Output:

```
   #
  ###
 #####
#######
```

Bài 2: Giải phương trình bậc 2

Bài 3: Giả sử 4 đồng có mệnh giá 1$, 2$, 5$, 10$. Nhập vào số tiền (nguyên) in ra số đồng tiền nhỏ nhất. Vd: 23$ = 2*10$+2$+1$, vậy cần 4 đồng

Bài 4: Tìm số lớn nhất và số bé nhất trong 1 dãy số

Bài 5: Tìm tổng và giá trị trung bình của 1 dãy số

Bài 6: Đếm số ký tự và từ xuất hiện trong chuỗi. Vd: "pay it forward" có 2 ký tự 'a' và 3 từ

Bài 7: Sắp xếp 1 dãy số có chiều dài n tăng dần, dùng thuật toán bubble sort

Bài 8: Sắp xếp 1 dãy số có chiều dài n giảm dần, dùng thuật toán selection sort

Bài 9: Tìm giá trị của e^2 dùng khai triển maclaurin tới bậc n = 10

Bài 10 : Viết chương trình encode và decode mã Caesar

https://en.wikipedia.org/wiki/Caesar_cipher

- Input: 1 chuỗi và 1 số
- Output: chuỗi đã được mã hóa và chuỗi sau khi giải mã

Bài 11, 12, 13 ưu tiên dùng struct và viết hàm (dùng hàm trả về struct hoặc dùng tham chiếu)

Bài 11: **C**ộng, trừ, nhân 2 số phức

Bài 12: **C**ộng, trừ, nhân 2 ma trận

Bài 13: **C**huyển vị ma trận

Bài 14: Viết định nghĩa và các hàm thao tác với stack. Gợi ý:

- Stack hoạt động theo cơ chế LIFO (last in first out)
- Struct của stack bao gồm: mảng chứa dữ liệu, vị trí đầu của stack.
- Hàm isEmpty kiểm tra có dữ liệu trong stack không, trả về true nếu stack rỗng
- Hàm isFull kiểm tra stack đã đầy chưa, trả về true nếu stack đầy
- Hàm push đưa dữ liệu vào stack
- Hàm pop lấy dữ liệu ra khỏi stack

Bài 15: Đọc và cho biết các thông số của file .wav. Gợi ý:

```
FILE *fptr      // Con trỏ kết nối với file cần thao tác
```

```
fptr = fopen("sample.wav","r");  // Mở file
fread(&data,sizeof(data),1,fptr);// Đọc file và lưu vào biến data
fclose(fptr);    // Đóng file
```

- 44 bytes đầu tiên chưa thông tin của file (header)
- Biến data có thể khai báo dùng array hoặc struct.
- Thông số cần quan tâm: SampleRate, BitsPerSample, NumChannels