\*\*\*\*\* 1 Byte là 1 ô nhớ \*\*\*\*\*\*

C Introduction, Data Types and Variables

Variables: Tên định danh của 1 ô/vùng nhớ dùng để lưu trữ dữ liệu. Biến có 3 loại chính là global, local, static

gọi tên : lấy giá trị

& + tên biến: lấy địa chỉ của biến đó

\* + tên biến: lấy giá trị của biến đó (giống gọi tên bình thường)

Data Type: Kiểu dữ liệu của biến như float, double, int, char, pointer (con trỏ), array, enum,...

Số nguyên: char, int, short, long, sizeof

char: 1 byte = 08 bit = range 256 = sign -128 ~ 127 = unsign 0 ~ 255

short: 2 byte = 16 bit = range 65,536 = sign -32,768 ~ 32,767 = unsign 0 ~ 65,535

int: 4 byte =32 bit = range 4,294,967,296 = sign -2,147,483,648~2,147,483,647 = unsign 0~4,294,967,295

pointer: 4 byte/8 byte (32/62 bits)

Số thực: float, double

float: 4 byte = 32 bit

double: 8 byte = 64 bit

Define using: struct, enum, logic

---------------------------------------------------------------------------

syntax:

variable:

"data\_type" "set\_name";

ex: int a;

ex: int b = 3;

pointer:

"data\_type"\* "set\_name";

ex: int a = 5;

int\* p = &a; // con trỏ p trỏ tới địa chỉ của a

ex: int\* p = (int\*)0xcabb2006; // con trỏ p trỏ có địa chỉ 0xcabb2006

ex: int a = 300; // 00000000 00000000 00000001 00101100

char \*p = &a; //lúc này giá trị của \*p trỏ tới không phải là 300

//mà là 44 ( 00101100 )

// bởi vì char \*p chỉ khai báo con trỏ p có 1 byte thôi

Pointer arithmetic: +, -, ++, --

int a = 500014; // 00000000 00000111 10100001 00101110

// giả sử địa chỉ các ô nhớ của a: 0x12345ab3 0x12345ab2 0x12345ab1 0x12345ab0

char \*p1 = &a; // lúc này p1 sẽ trỏ tới địa chỉ 0x12345ab0

p1 = p1 + 1; // lúc này p1 sẽ trỏ tới địa chỉ 0x12345ab1

short \*p2 = &a;// lúc này p2 trỏ tới đ.chỉ 0x12345ab1 và 0x12345ab0 => lấy giá trị 10100001 00101110

p2 = p2 + 1; // lúc này p2 trỏ tới đ.chỉ 0x12345ab3 và 0x12345ab2 => lấy giá trị 00000000 00000111

==>> step\_size con trỏ phụ thuộc vào kiểu dữ liệu gọi

ví dụ (nâng cao): giống ví dụ trên, nhưng muốn lấy giá trị tại 0x12345ab2 0x12345ab1

int a = 500014; // 00000000 00000111 10100001 00101110

// giả sử địa chỉ các ô nhớ của a: 0x12345ab3 0x12345ab2 0x12345ab1 0x12345ab0

char \*p1 = &a; // lúc này p1 sẽ trỏ tới địa chỉ 0x12345ab0

p1 = p1 + 1; // lúc này p1 sẽ trỏ tới địa chỉ 0x12345ab1

short \*p2; // lúc này p2 sẽ trỏ tới địa chỉ random 2 byte

p2 = p1; // lúc này p2 sẽ trỏ tới địa chỉ 0x12345ab2 và 0x12345ab1 => lấy giá trị 00000111 10100001

Array và Pointer bản chất là 1

ví dụ: int arr[] = {1, 2, 3, 4};

int\* p

<=> cái tên arr[] thực chất là \*p (diễn đạt khác nhau thôi)

khi ta gọi arr[2]; kết quả sẽ là 3

p = arr[]; p = p + 2; thì kết quả cũng là 3

---------------------------------------------------------------------------

const: constance là hằng số không bị thay đổi (giống define)

define không tốn bộ nhớ

const lưu tên biến bằng 1 bộ nhớ

symtag: const <data\_type> <var\_name\_set> = <value>;

vd: const int a = 22;

\*\*note: void test(const int\* arr) => có const thì chỉ read-only, không thể thay đổi

#include <stdio.h>

void example(const char\* str){

\*str = 6;

}

int main(){

char str[5];

example(str);

}

bình thường nếu không có const, chương trình này sẽ chạy bình thường

vì có const (read-only) nên \*str = giá trị mới sẽ bị lỗi

. pointer to const: trỏ tới giá trị không đổi

symtag: const <data\_type> <var\_name\_set> = <value>;

vd: const int a = 22;

. const pointer: con trỏ có địa chỉ không thay đổi (giá trị của ô nhớ tại const pointer vẫn thay đổi được)

symtag: <data\_type>\* const <var\_name\_set> = <địa chỉ ô nhớ>;

int\* const p = &a;

----------------------------------------------------------

Memory: chia các vùng nhớ do file linker quyết định

text (code) read-only

data (RW) global\_var đã được khởi tạo (!= NULL != 0)

static đã được khởi tạo (!= NULL != 0)

bss (Zi) global\_var chưa được khởi tạo giá trị (== NULL) hoặc đã khởi tạo = 0

static chưa được khởi tạo (!= NULL != 0)

heap cấp phát động (dành cho lập trình quản lí) các hàm sử dụng để tác động:

malloc xin vùng nhớ (không reset giá trị của vùng nhớ)

calloc xin vùng nhớ (Reset giá trị vùng nhớ về 0)

realloc Resize lại vùng nhớ đã xin

free trả lại vùng nhớ đã xin

stack local\_var

parametter của hàm

struct,…

+ biến cục bộ (local / auto): là biến mình khai báo bình thường, chỉ tồn tại nội bộ trong {}, ra khỏi {} sẽ tự clear

+ biến static: là biến có khai báo có thêm chữ <static> ở đằng trước. vd: int static a = 6;

là biến giống như auto nhưng không bị xóa (chỉ sử dụng nội bộ ở 1 hàm nào đó, kết thúc hàm đó giá trị không bị xóa đi)

+ biến toàn cục (global): là biến ngoài {}: là biến khai báo ngoài không thuộc bất kì hàm nào, tồn tại xuyên suốt toàn chương trình (các hàm khác nhau đều có thể sử dụng chung và không bị xóa đi)

\* static global variable: chỉ xài trong 1 file .c cố định

\*\* static: hạn chế phạm vi sử dụng trong 1 file .c cố định vd: static int a;

\*\* extern: biến sử dụng chung cho nhiều file khác nhau vd: extern int a;

\*\* volatile: không cho optimize chương trình. vd: volatile int a;

--------------------------------------

Macro: define là đặt nickname đại diện cho một đối tượng nào đó ---- không tốn dung lượng bộ nhớ

symtax: #define <name\_set> <giá trị mong muốn name\_set làm đại diện>

vd: #define AB 10

---------------------------------------------------------------------------------------------------------------------

struct (user defind)

symtax: declare: template: không tốn bộ nhớ

step 1:

struct <name\_struct\_set>{

<data\_type> <name\_variable\_set>;

....

};

gọi hàm struct:

struct <name\_struct> <name\_var\_set>;

truy cập vào thì sử dụng toán tử "dot"

con trỏ struct truy cập = dấu "->" vd: struct test a;

struct\* b = &a;

b->a.dummy1 = value;

\*\*vd 1: #include <stdio.h>

struct test\_struct{ //phần này lúc khởi tạo chưa tốn dung lượng bộ nhớ

int test\_int;

char test\_char;

short int test\_short;

char dummy1;

};

int main(){

struct test\_struct sinhvien; //khởi tạo này đã tốn dung lượng bộ nhớ bằng tổng các dung lượng thành phần

sinhvien.test\_char = 't'; //(int 4byte, char 1byte, short 1byte, nên thêm 1 dummy char 1byte để tổng chia hết cho 4)

sinhvien.test\_int = 1; // a được hiểu như là đối tượng

sinhvien.test\_short = 22;

printf("char: %c\nint: %d\nshort: %d\n", sinhvien.test\_char, sinhvien.test\_int, sinhvien.test\_short);

return 0;

}

\*\*vd 2:

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

////////--------(1)[

typedef struct diem{

int diemtoan;

int diemly;

} diem\_typedef;

typedef struct sinhvien{ //struct trong struct

char name[30];

int mssv;

diem\_typedef diem; //thay vi nhap tung diem ly diem toan, ta dung 1 struct khac quan li diem

} sinhvien\_t;

////////--------(1)]

////////--------(2)[

typedef struct s\_max\_min{

int max;

int min;

} a\_typedef; //a\_typedef la ten duoc dat lai

struct s\_max\_min s\_find\_max\_min(int arr[], int sophantu){

struct s\_max\_min test; // tao mot struct local <=> var\_local

test.max = 0;

test.min = pow(2, 31) - 1;

for (short i=0; i<sophantu; i++){ //nhap gia tri cho mang

arr[i] = rand()%100;

}

printf("mang duoc nhap la: \n"); //in ra de xem mang vua nhap

for (short i=0; i<sophantu; i++){

printf("%d\t", arr[i]);

}

printf("\n");

for (short i=0; i<sophantu; i++){

if (arr[i] > test.max){

test.max = arr[i];

}

if (arr[i] < test.min){

test.min = arr[i];

}

}

return test;

}

////////--------(2)]

////////--------(3)[

void find\_max\_min(int arr[], int sophantu, int\* max, int\* min){

\*max = 0;

\*min = pow(2, 31)-1;

for (short i=0; i<sophantu; i++){ //nhap gia tri cho mang

arr[i] = rand()%100;

}

printf("mang duoc nhap la: \n"); //in ra de xem mang vua nhap

for (short i=0; i<sophantu; i++){

printf("%d\t", arr[i]);

}

printf("\n");

for (short i=0; i<sophantu; i++){

if (arr[i] > \*max){

\*max = arr[i];

}

if (arr[i] < \*min){

\*min = arr[i];

}

}

}

////////--------(3)]

////////--------(4)[

typedef struct date{ //bit field

// unsigned int day;

// unsigned int month;

// unsigned int year;

unsigned int day:5; //bien int day co 4 byte; nhung :5 co nghia chi can xai 5 bit

unsigned int month:4;//bien int month co 4 byte; nhung :4 co nghia chi can xai 4 bit

unsigned int year:12;//bien int year co 4 byte; nhung :12 co nghia chi can xai 12 bit

} date\_mont\_year;

////////--------(4)]

int main(){

int arr[10]; //khai bao mang gom 10 phan tu

int n; //so phan tu cua mang

int max = 0;

int min = 0;

a\_typedef test\_c2 = {5, 6}; //cach 2: khai bao nhu khai bao mang

printf("cach 2: %d\t%d\n", test\_c2.max, test\_c2.min);

a\_typedef test\_c3 = {.max = 10, .min = 0}; //cach 3: khai bao nhu khai bao mang ma ro rang hon

printf("cach 3: %d\t%d\n", test\_c3.max, test\_c3.min);

a\_typedef find\_var; //cach 1: khai bao doi tuong find var

find\_var.max = 0;

find\_var.min = 1;

printf("nhap so phan tu cua mang: ");

scanf("%d", &n);

find\_max\_min(arr, n, &max , &min);

printf("max: %d\tmin: %d\n", max , min );

find\_var = s\_find\_max\_min(arr, n);

printf("%d\t%d\n", find\_var.max, find\_var.min);

sinhvien\_t sv1;

sv1.mssv = 17151155;

sv1.diem.diemly = 9;

sv1.diem.diemtoan = 10;

printf("\nmssv: %d\tdiem ly: %d\tdiem toan: %d\n", sv1.mssv, sv1.diem.diemly, sv1.diem.diemtoan);

date\_mont\_year date = {8, 12, 2023};

printf("\nday: %d\tmonth: %d\tyear: %d\n", date.day, date.month, date.year);

printf("size struct date: %d\n", sizeof(date)); //12 byte neu khong xai bit field

//do xai bit filed (5+4+12 = 19bit <32bit) => chi ton 4 byte

return 0;

}

------------------------------------------------------------------------------------------------------------------------

typedef: cũng như define, không tốn dung lượng bộ nhớ

(user defind)

định nghĩa lại cho kiểu dữ liệu đã có rồi

symtax: typedef <existing\_name> <alias\_name> vd: typedef int INT32;

\*\*ứng dụng với struct: ví dụ:

struct test{ typedef struct test{

..... ....

}; } test\_t;

khi gọi hàm ta dùng lúc này khi gọi hàm ta chỉ

struct test thì gọi dài quá cần gọi test\_t thay cho cụm struct test

------------------------------------------------------------------------------------------------------------------------

bit field: (giống struct nhưng có cụ thể dung lượng sử dụng -->> dùng để tiết kiệm bộ nhớ)

symtax: declare: template: không tốn bộ nhớ

step 1:

struct <name\_struct\_set>{

<data\_type> <name\_variable\_set>:<số bit cần sử dụng>;

....

};

gọi hàm struct:

struct <name\_struct> <name\_var\_set>;

vd:

typedef struct date{ //bit field

// unsigned int day;

// unsigned int month;

// unsigned int year;

unsigned int day:5; //bien int day co 4 byte; nhung :5 co nghia chi can xai 5 bit

unsigned int month:4;//bien int month co 4 byte; nhung :4 co nghia chi can xai 4 bit

unsigned int year:12;//bien int year co 4 byte; nhung :12 co nghia chi can xai 12 bit

} date\_mont\_year;

-------------------------------------------------------------------------------------------------------------------------

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#pragma pack(1) //defaul se la 4/8 (tuy 32/64 bit) -> lenh nay dinh nghia lai kich thuoc min = 1

typedef struct{

int a;

int b;

char c;

char dummy;

long long d;

}struct\_a;

int main(){

struct\_a str;

str.a = 2;

str.b = 4;

//con tro trong struct

struct\_a\* p\_str; //con tro p\_str tro toi struct\_a, luu o stack\_memory, co kich thuoc =4/8byte (tùy 32/64 bit)

p\_str = &str;

//#define address\_str 0xff00ff00 //su dung trong truong hop biet truoc dia chi bat dau cua str

//#define p\_str (struct\_a\*)address\_str

//truy cap struct bang con tro (cach 1)

p\_str -> a = 7; //truy cap. sua lai gia tri cua str.a (2 doi thanh 7)

//truy cap struct bang con tro (cach 2)

(\*p\_str).b = 9;//truy cap. sua lai gia tri cua str.b (4 doi thanh 9)

printf("a: %d\t b: %d\n---------------\n\n", str.a, str.b);

//defaul: kich thuoc struct se alliment theo kieu du lieu lon nhat co trong struct

//trong TH nay thi longlong\_type co size 8 byte la lon nhat nen lay theo 4x2 + 8 + 8 = 24byte

//(4x2 la do cung kieu int nen gop chung duoc 2 int = 8 byte)

printf("size of struct: %d\n", sizeof(str)); //kq: 24 byte ( 4x2 + 8 + 8 = 24 )

//kq: 18 byte ( 4 + 4 + 1 + 1 + 8 = 18 ) (using #pragma pack(1))

//khong de size theo defaul, dung lenh <#pragma pack(2^n)> voi 2^n la size mong muon (byte)

//dung #pragma pack(2^n) de tiet kiem bo nho, nhung khien toc do xu li bi cham di (can nhac su dung cho tung T.H)

//neu defaul, de tiet kiem bo nho, ta dat cac type giong nhau o sat nhau

//(vi du: int, int, char, char, long long (24byte) <> int, int, char, long long, char (32 byte) )

//thong nhat la dat. theo thu tu tang dan hoac giam dan cua kich thuoc data\_type

return 0;

}

-------------------------------------------------------------------------------------------------------------------

enum:

là 1 loại macro, nhưng lại thuộc constance (tốn dung lượng khi khai báo và có giá trị cố định)

(giống struct nhưng giá trị bên trong không thay đôi khi đã khai báo)

Kích thước mặc định của enum luôn là 4 byte (dù 32/64 bit)

Hoặc nó sẽ lấy giá trị theo data\_type lớn nhất (nếu có data vượt qua 4 byte như kiểu long long)

symtax: enum <enum\_name\_set>{

<thành phần 1>,

<thành phần 2>,

...,

<thành phần n>

};

vd:

enum date\_on\_week{ //khi khai báo mặc định như thế này, số mặc định ban đầu sẽ là 0 và cứ thế + lên

thu2, //0

thu3, //1

thu4, //2

thu5, //3

thu6, //4

thu7, //5

cn //6

};

int main(){

enum date\_on\_week date;

date = thu2;

printf("%d", date); //kq = 0

return 0;

}

vd:

enum date\_on\_week{ //khi khai báo mặc định như thế này, số mặc định ban đầu sẽ là 0 và cứ thế + lên

thu2 = 2, //2

thu3, //3 //Lưu ý: không khai báo giá trị giữa chừng

thu4, //4 //ví dụ: ta đã khai báo thu2 = 2 thì các thu3, 4,... không khai báo giá trị

thu5, //5 // vì như vậy sẽ làm mất ý nghĩa của enum

thu6, //6

thu7, //7 // giả sử không khai báo thu2 = 2 mà khai báo từ thu4 = 4 thì vẫn được

cn //8 // lúc này thu2=0, thu3=1; thu4=4, thu5=5....

};

int main(){

enum date\_on\_week date;

date = thu2;

printf("%d", date); //kq = 2

return 0;

}

\*\*có thể kết hợp với typedef giống struct

-----------------------------------------------------------------------------------------------------------

union: bản chất giống hệt struct nhưng khác ở chỗ

struct sử dụng nhiều biến thành viên được

union chỉ sử dụng được duy nhất 1 biến thành viên, muốn sử dụng lại thì lại phải khởi tạo lại

do đó có kích thước của data\_type thành viên lớn nhất

symtax: union <union\_name\_set>{

<date\_type> <data\_name\_set\_1>;

<date\_type> <data\_name\_set\_2>;

...

<date\_type> <data\_name\_set\_n>;

};

ví dụ:

union name\_uni{

short a;

int b;

long long c;

};

int main(){

int check;

union name\_uni uni1;

uni1.a = 5;

uni1.b = 10;

printf("%d\n", uni1.a); //kq = 10 <=> do no chi su dung duoc 1 cai cuoi cung duoc khai bao

union name\_uni uni2;

uni2.a = 5;

printf("%d\n", uni2.a); //kq = 5

printf("%d", sizeof(uni1)); //kq = 8 <=> lay theo gia tri kich thuoc cua data\_type lon nhat

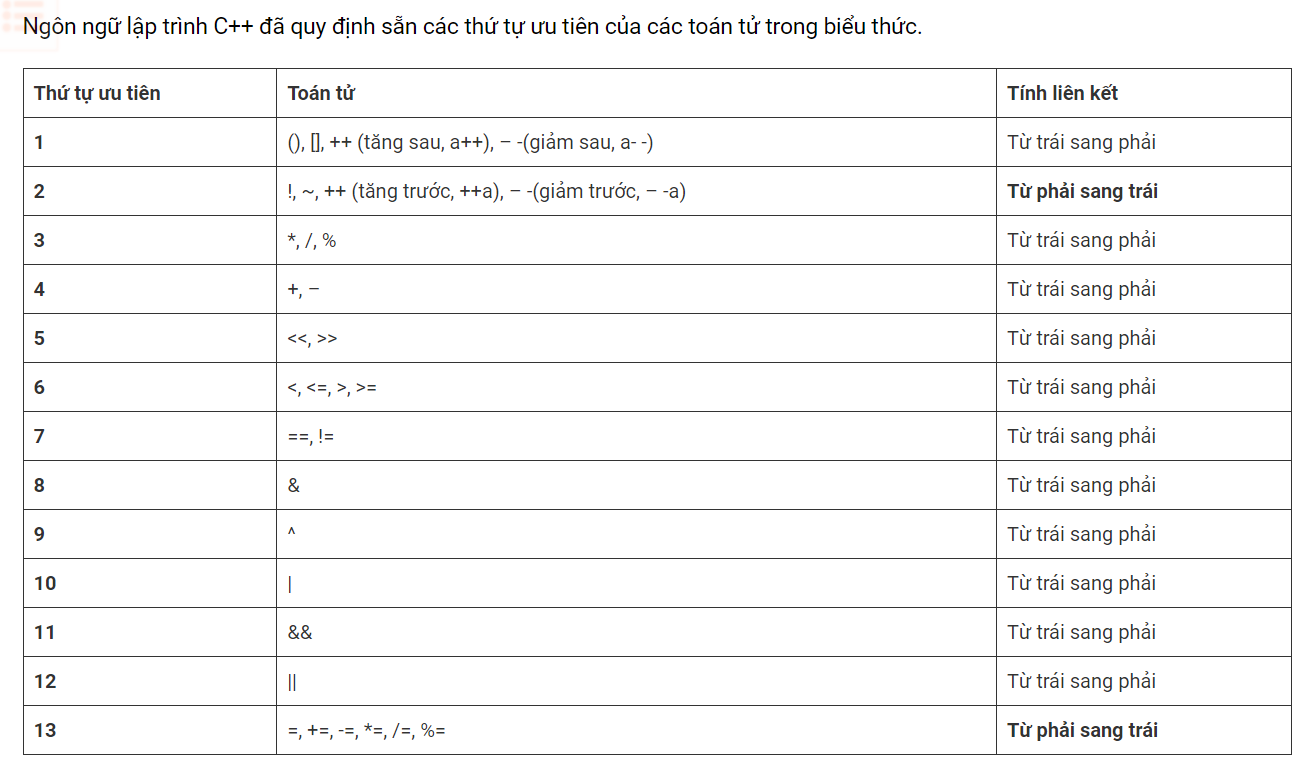
return 0;

}

--------------------------------------------------------------------------------------------------------------

1 số lưu ý khác:

- Thứ tự ưu tiên các phép toán



- Bảng ASCII:

