



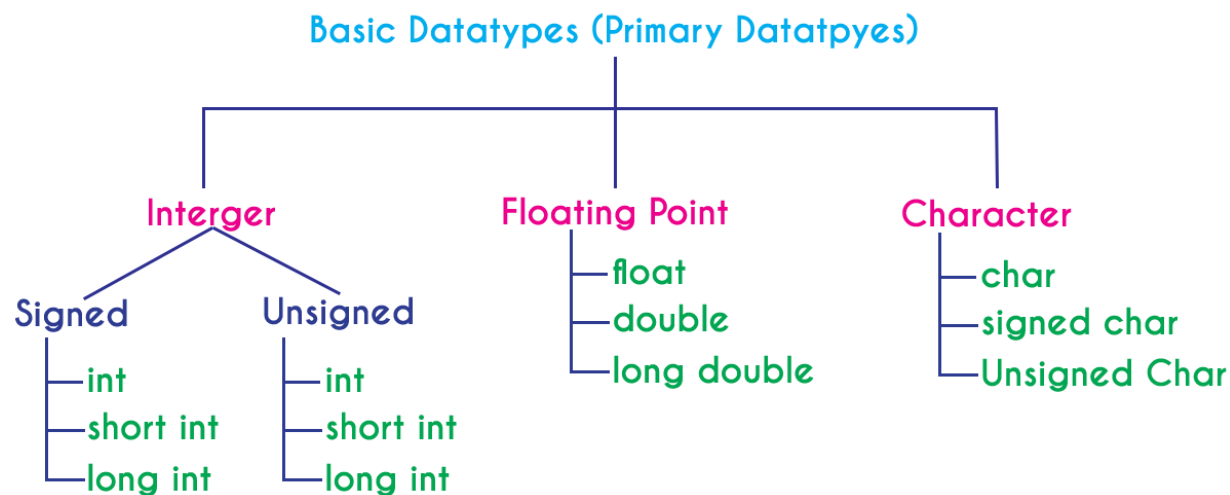
# Ôn lại NNLT

ONE LOVE. ONE FUTURE.

# Biến (variable), kiểu (type), giá trị (value)

- **Biến** : đại lượng được lưu trữ trong bộ nhớ = khối nhớ
- Chứa **giá trị**, có thể thay đổi trong khi chạy CT
- Kích thước khối nhớ tương ứng với **kiểu** của biến
- Biến cần được khai báo trước khi dùng: **kiểu tên;**
- Phạm vi toàn cục hoặc cục bộ.

# Các kiểu dữ liệu cơ bản



Type	Size (Bytes)	Range
int (signed short int)	2, 4, 8	depends...
short int (signed short int)	2	-32768 to +32767
long int (signed long int)	4	-2,147,483,648 to +2,147,483,647
unsigned int (unsigned short int)	2, 4, 8	depends...

Type	Size (Bytes)	Range
float	4	1.2E - 38 to 3.4E + 38
double	8	2.3E-308 to 1.7E+308
long double	10	3.4E-4932 to 1.1E+4932

125 (cs 10) = 0175 (cs 8) = 0x7D (cs 16) ;

150 (cs 10) = 0226 (cs 8) = 0X96 (cs 16) ;

# Các kiểu dữ liệu cơ bản

Type	Size (bytes)	Range
char (signed char)	1	-128 to +127
unsigned char	1	0 to 255

Biểu diễn hằng ký tự:

- 'a', 'A', '0', '.', ...
- '\t', '\n', '\", '\", '\0', ...
- '\x49', '\l', 73, 0x49

<ctype.h>: *toASCII(c)*

(American Standard Code for Information Interchange)

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
32	Space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	\$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u

- enum:

- `enum Animal { Cat, Dog, Tiger, Lion };`
- `enum dayInWeek {Mo, Tu, We, Th, Fr, Sa, Su};`

- struct:

```
typedef struct {  
    char name[20];  
    unsigned int age;  
    enum {Nam, Nu} gender;  
    struct {  
        char city[20];  
        char street[20];  
        int number;  
    } address;  
} Student;
```

- Câu lệnh được dùng để thực hiện một nhiệm vụ trong chương trình: gán, tính toán, đọc/ghi dữ liệu, gọi hàm,...
- Phân loại:
  - Câu lệnh đơn
    - `printf("Xin chao!");`
    - `x = PI*R*R;`
  - Các lệnh rẽ nhánh và điều kiện: *for, if, while, switch*,...
  - Đặc biệt: câu lệnh trống, câu lệnh biểu thức, tạo nhãn, *return*,...
  - Khối lệnh: Gộp các lệnh đơn lại bằng `{ ... }`

# Câu lệnh

```
if ( condition )
{
    ....
    True block of statements;
    ....
}
else
{
    ....
    False block of statements;
    ....
}
```

```
switch ( expression or value )
{
    case value1: set of statements;
    ....
    case value2: set of statements;
    ....
    case value3: set of statements;
    ....
    case value4: set of statements;
    ....
    case value5: set of statements;
    ....
    .
    .
    default: set of statements;
}
```


**break;**

```
while( condition )
{
    ...
    block of statements;
    ...
}
```


```
do
{
    ...
    block of statements;
    ...
} while( condition );
```

```
for( initialization ; condition ; modification )
{
    ...
    block of statements;
    ...
}
```


```
for (initilization; condition; modification)
{
    ....
    break ;
    ....
}
```




```
while ( condition)
{
    ....
    break ;
    ....
}
```



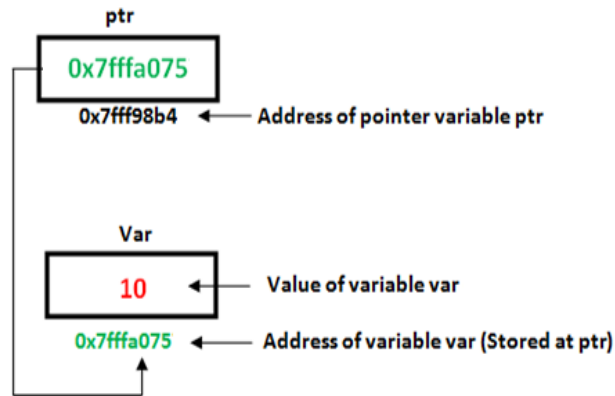
```
for (initilization; condition; modification)
{
    ....
    continue;
    ....
}
```



```
while ( condition)
{
    ....
    continue;
    ....
}
```



- Biến con trỏ là một biến đặc biệt dùng để chứa địa chỉ vùng nhớ của một biến khác.



- Khai báo 1 biến con trỏ: toán tử '\*'
  - `int *ip;    int* ip;        int * ip;`
  - `double *dp;`
  - `char *ch`
  - `int *pi, j;`
  - `int* pi, j;`
  - `int *pi, *j;`
- Kích thước của con trỏ tương đương kích thước của `int`

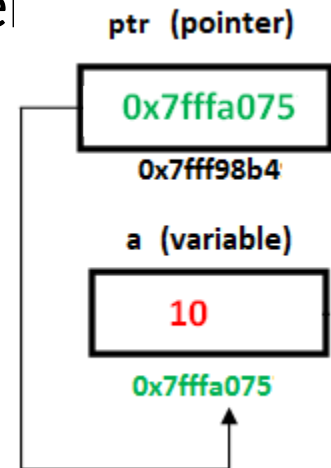


- Toán tử địa chỉ & dùng để lấy địa chỉ của một biến
- Gán địa chỉ mới cho con trỏ

```
• int *pInt, *pInt2, iNT3;  
  pInt2 = pInt;  
  pInt2 = &iNT3;
```

- Truy xuất vùng nhớ con trỏ trỏ tới (toán tử \*):

```
• int aInt = *pInt;  
• *pChar = 'A';  
• char s[10] ;  
  char * ps = s ;  
  ps[0] = 'H' ;  
  ps[1] = 'I' ;  
  ps[2] = '\\0' ;
```



```
*a      // invalid  
*ptr    =10  
*&a     =10  
*&ptr   = 0x7ffa075  
&*a     // invalid  
&*ptr   = 0x7ffa075
```

& là toán tử ngược với \*, với một biến a bất kỳ thì \*&a tương đương với a, và nếu p là một con trỏ thì &\*p cũng tương đương với p

# Các phép toán với con trỏ

- Tăng/giảm: để thay đổi con trỏ trỏ tới vị trí tiếp theo (tương ứng với kích thước kiểu nó trỏ tới)

Địa chỉ	1500	1501	1502	1503	1504	1505	1506	1507	1508	1509	1510	1511
			p-- (1502)		short *p (1504)		p++ (1506)					

- Cộng địa chỉ: cũng tương ứng với kiểu nó trỏ tới

Địa chỉ	1500	1501	1502	1503	1504	1505	1506	1507	1508	1509	1510	1511
	p-2 (1500)				short *p (1504)						p+3 (1510)	

- So sánh: 2 con trỏ cùng kiểu có thể được so sánh địa chỉ với nhau như 2 số nguyên (lớn, nhỏ, bằng)
- Hai con trỏ cùng kiểu có thể trừ cho nhau để ra số phần tử sai khác

## Con trỏ void \*

- Là con trỏ nhưng không mang thông tin về kiểu
- Có thể được chuyển kiểu ngầm định sang bất kỳ kiểu con trỏ nào khác

```
int a = 10;  
char b = 'x';
```

```
void *p = &a; // void pointer holds address of int 'a'  
p = &b; // void pointer holds address of char 'b'
```

- Không dùng toán tử truy xuất giá trị \* với con trỏ void \*

```
#include<stdio.h>  
int main()  
{  
    int a = 10;  
    void *ptr = &a;  
    printf("%d", *ptr);  
    return 0;  
}
```

Compiler Error:

```
#include<stdio.h>  
int main()  
{  
    int a = 10;  
    void *ptr = &a;  
    printf("%d", *(int *)ptr);  
    return 0;  
}
```

10

- Con trỏ void \* được dùng để làm việc với bộ nhớ thuần túy hoặc để thao tác với những biến chưa xác định kiểu

# Hằng con trỏ, con trỏ tới hằng

- **const T \* ptr** = <exp>; // Con trỏ trỏ tới một hằng  
**T const \* ptr** = <exp>; // Con trỏ trỏ tới một hằng
- **T \* const ptr** = <exp>; // Hằng con trỏ trỏ tới một biến
- **const T \* const cons3** = <exp>; // Hằng con trỏ trỏ tới một hằng

```
const char* pc= "abcd" ;  
char * s = "fgh" ;  
pc[1] = 'f'; //error  
pc = "efgh"; //OK  
pc = s ; //OK
```

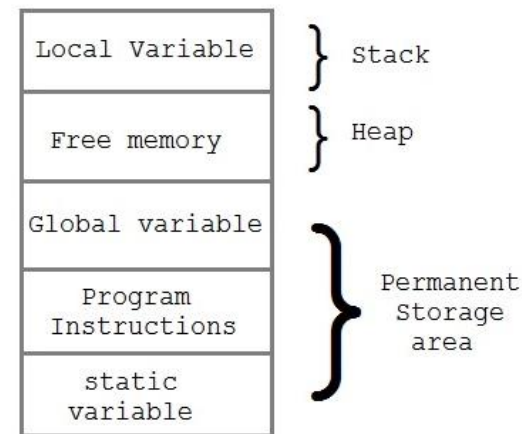
```
const char* const cpc = "abcd";  
cpc[1] = 'f' ; //error  
cpc = "efgh"; //error  
cpc = s ; //error
```

```
char* const cp = "abcd";  
cp[1] = 'f' ; //OK  
cp = "efgh"; //error  
cp = s ; //error
```

- Là một hằng con trỏ mang ý nghĩa đặc biệt là không trỏ tới địa chỉ nào trong bộ nhớ.
- Tùy vào thư viện chuẩn, con trỏ Null thường = 0.
- Không được gán giá trị cho con trỏ NULL
  - `int *pInt = NULL`  
`*pInt = 100; /* lỗi */`
- Cần phân biệt con trỏ NULL và con trỏ chưa được khởi tạo (trỏ đến địa chỉ ngẫu nhiên)
  - → để tránh lỗi, luôn gán con trỏ bằng NULL khi chưa hoặc tạm thời không được dùng tới
- Xác định tính hợp lệ của một biến con trỏ.
  - `if (p != NULL)`
  - `if (p)`
  - `if (!p)`

# Cấp phát bộ nhớ động (Dynamic Memory Allocation)

- Các biến khai báo được tạo ra và cấp phát bộ nhớ khi khai báo (trong stack)
- Có khi cần cấp phát theo nhu cầu sử dụng mà không biết từ khi viết chương trình → cấp phát động (trong heap)



- Cấp phát bộ nhớ:

- `#include <stdlib.h>`
- `void *malloc(int size)`
- `int *p = (int *)malloc(10*sizeof(int));`
- `void *calloc(int num_elem, int elem_size)`
- `void *realloc(void* ptr, int new_size)`
- Việc cấp phát có thể không thành công và trả về NULL → cần kiểm tra

- Huỷ (trả lại) vùng nhớ đã được cấp phát:

- `void free(void *p);`
- `free(p);`

- Hàm là một khối các câu lệnh thực hiện một nhiệm vụ nhất định, và có thể được gọi khi cần
- Hàm xác định bởi **tên, tham số, kiểu giá trị trả về**
- Khai báo hàm: Sử dụng nguyên mẫu
  - <kiểu trả về> <tên hàm>(<khai báo các tham số>);
- Định nghĩa hàm:
  - <kiểu trả về> <tên hàm>(<khai báo các tham số>) {  
    *Khai báo các biến dùng cho hàm*  
    *Các câu lệnh của hàm*  
}
- Gọi hàm:
  - <tên hàm>(<các tham số thực>)
  - biến = <tên hàm>(<khai báo các tham số>);
- Lệnh return:
  - Kết thúc hàm và trả về một giá trị cho nơi gọi nó
- Kiểu trả về = void: Hàm không trả về giá trị gì

- Ví dụ:

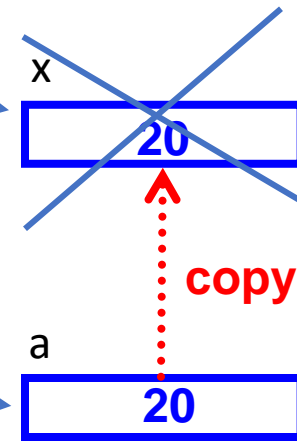
- ```
double tong(double x, double y);  
double tich(double x, double y);  
int main() {  
    double x = 5., y = 10.;  
    tong(x, y);  
    tich(x, y);  
    return 0;  
}  
  
double tong(double x, double y) { return x+y; }  
double tich(double x, double y) { return x*y; }
```



# Tham số kiểu giá trị và kiểu con trỏ (tham chiếu)

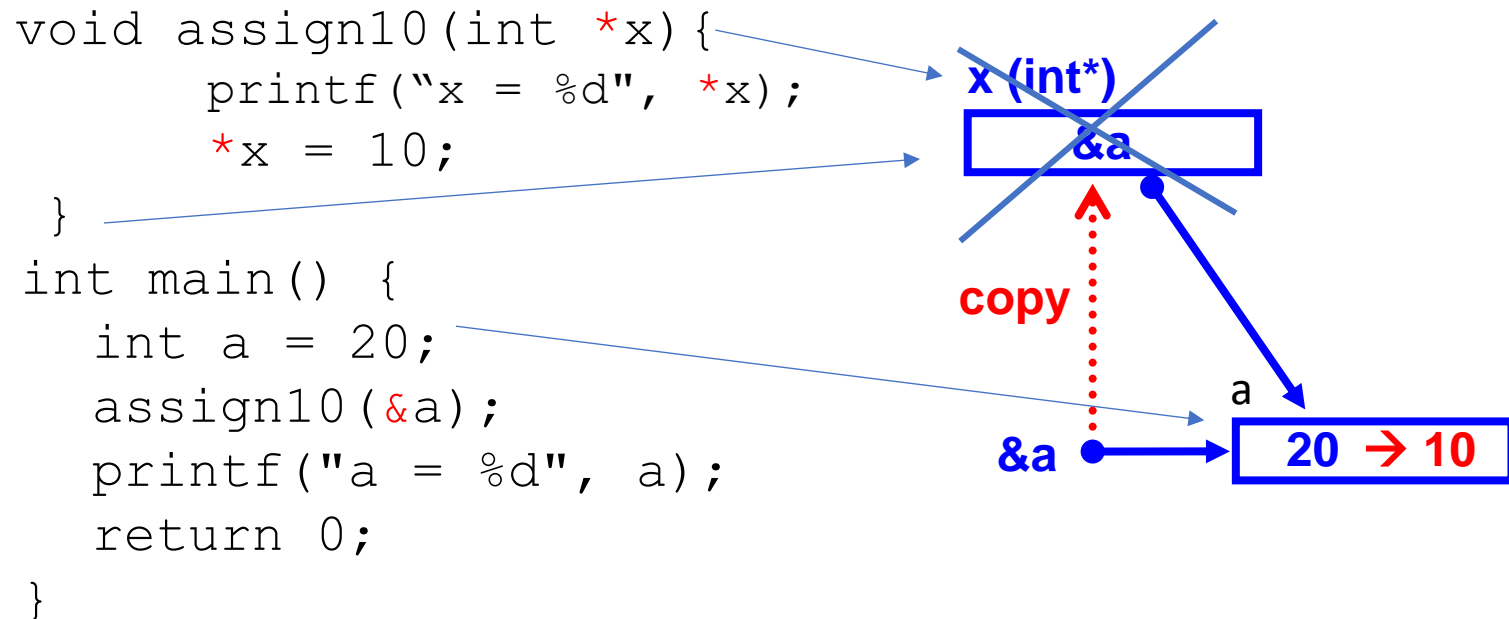
- Tham số của hàm là biến tạm thời, tạo ra khi gọi và huỷ khi hàm kết thúc  
→ gán giá trị cho tham số không ảnh hưởng tới biến nơi gọi.

```
• void assign10(int x) {  
    printf("x = %d", x);  
    x = 10;  
}  
int main() {  
    int a = 20;  
    assign10(a);  
    printf("a = %d", a);  
    return 0;  
}
```



# Tham số kiểu giá trị và kiểu con trỏ (tham chiếu)

- Dùng con trỏ nếu muốn thay đổi giá trị của biến ở nơi gọi (tham chiếu tới biến ở nơi gọi)



- Tham số con trỏ thường được dùng như một cách khác để trả về thêm giá trị, vì mỗi hàm chỉ có một giá trị trả về theo đúng nghĩa

- Trả về giá trị thông qua tham số kiểu con trỏ

```
float sum(float a[], int N){  
    int i;  
    float sf=0;  
    for (i=0;i<N;i++) sf += a[i];  
    return sf;  
}
```

```
void sum(float a[], int N, float* s){  
    int i;  
    float sf=0;  
    for (i=0;i<N;i++) sf += a[i];  
    *s = sf;  
}
```

- Biến toàn cục: được khai báo ở ngoài các hàm, có phạm vi trong toàn chương trình và tồn tại trong suốt quá trình chạy.
  - Được truy cập bởi tất cả các hàm định nghĩa sau khai báo biến toàn cục.
- Biến địa phương: được khai báo ở trong một hàm hoặc một khối lệnh, chỉ có phạm vi trong hàm/khối đó, và bị huỷ sau khi kết thúc chạy hàm/khối đó
  - Các khai báo tham số và các biến nội bộ của một hàm chỉ giới hạn trong phạm vi của hàm đó
  - Khai báo biến địa phương sẽ “che” mất biến cùng tên khác có phạm vi rộng hơn

```
int x = 10, y = 20;
int sum(int x, int y) {
    return x+y;
}
int main() {
    int x = 100, y = 200;
    {
        int x = 1, y = 2;
    }
    int z = sum(x, y);
    return 0;
}
```

- Là biến chỉ có phạm vi địa phương nhưng vẫn tồn tại ngay cả khi chưa vào hoặc đã thoát khỏi hàm/khối
- Khai báo bằng cách thêm từ khoá static
  - ```
int callCount() {  
    static int count = 0;  
    count++;  
    return count;  
}
```
- Biến static chỉ được khởi tạo 1 lần. Nếu không khởi tạo, tự gán =0
- Cũng có biến static toàn cục: thuộc nội bộ của một file nguồn
  - ```
static int tic_time = 0;  
void tic() {  
    tic_time = clock();  
}  
int toc() {  
    return clock() - tic_time;  
}
```

