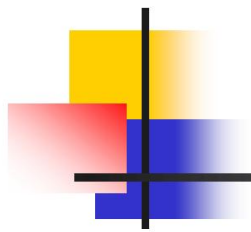


Con trỏ

Chỉ dành cho Trung tâm Aptech sử dụng

Phiên 8



Mục tiêu

Giải thích con trỏ là gì và được sử dụng ở đâu

Giải thích cách sử dụng biến con trỏ và toán tử
con trỏ

Chỉ dành cho Trung tâm Aptech sử dụng
Gán giá trị cho con trỏ

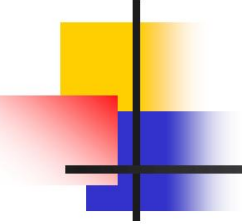
Giải thích phép tính số

học con trỏ Giải thích so

sánh con trỏ Giải thích con trỏ và mảng một

chiều Giải thích con trỏ và mảng nhiều chiều

Giải thích cách phân bổ bộ nhớ diễn ra



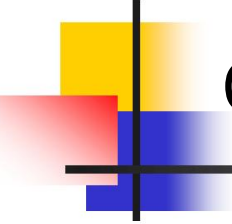
Con trỏ là gì?

Con trỏ là một biến chứa địa chỉ của một vị trí bộ nhớ của một biến khác

Nếu một biến chứa địa chỉ của một biến khác biến, biến đầu tiên được cho là trỏ đến biến thứ hai

Con trỏ cung cấp phương pháp gián tiếp để truy cập giá trị của một mục dữ liệu

Con trỏ có thể trỏ đến các biến có kiểu dữ liệu cơ bản khác như int, char hoặc double hoặc dữ liệu tổng hợp như mảng hoặc cấu trúc



Con trỏ được sử dụng để làm gì?

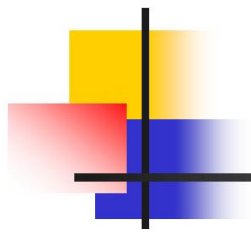
Một số tình huống có thể sử dụng con trỏ là -

- Để trả về nhiều hơn một giá trị từ một hàm

- Chỉ dành cho Trung tâm Aptech sử dụng
Để truyền các mảng và chuỗi thuận tiện hơn từ hàm này sang hàm khác

- Để thao tác các mảng một cách dễ dàng bằng cách di chuyển các con trỏ đến chúng thay vì di chuyển chính các mảng

- Phân bổ bộ nhớ và truy cập bộ nhớ
(Phân bổ bộ nhớ trực tiếp)



Biến con trỏ

Một khai báo con trỏ bao gồm một kiểu cơ sở và một tên biến được đặt trước bởi một *

Chỉ dành cho Trung tâm Aptech sử dụng

Cú pháp khai báo chung là:

```
gõ *tên;
```

Ví dụ:

```
int *var2;
```



Toán tử con trỏ

Có 2 toán tử đặc biệt được sử dụng với con trỏ

& Và *

Toán tử **&** là toán tử một ngôi và nó trả về địa chỉ bộ nhớ của toán hạng

biến2 = &var1;

Toán tử thứ hai ***** là phần bù của **&**. Đây là toán tử một ngôi và trả về giá trị chứa trong vị trí bộ nhớ được trỏ tới bởi giá trị của biến con trỏ

nhiệt độ = *var2;

Gán giá trị cho Con trỏ-1

Giá trị có thể được gán cho con trỏ thông qua toán tử & .

```
ptr_var = &var;
```

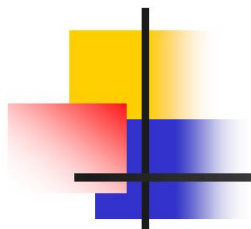
Ở đây địa chỉ của `var` được lưu trữ trong biến `ptr_var`

Cũng có thể gán giá trị cho con trỏ thông qua một biến con trỏ khác trỏ đến một mục dữ liệu có cùng kiểu dữ liệu

```
ptr_var = &var;
```

```
ptr_var2 = ptr_var;
```

Gán giá trị cho Con trỏ-2



Các biến có thể được gán giá trị thông qua các con trỏ của chúng như
Tốt

Chỉ dành cho Trung tâm Aptech sử dụng

```
*ptr_var = 10;
```

Khai báo trên sẽ gán 10 cho biến var nếu
ptr_var trỏ đến var



Số học con trỏ-1

Phép cộng và phép trừ là những phép toán duy nhất có thể được thực hiện trên con trỏ

```
int var, *ptr_var;
```

Chỉ dành cho các nhà lập trình C sử dụng

```
ptr_var = &var;
```

```
var = 500;
```

```
ptr_var++;
```

Giả sử var được lưu trữ tại địa chỉ
1000

Sau đó ptr_var có giá trị 1000 được lưu trữ trong đó. Vì số nguyên dài 2 byte, sau biểu thức "ptr_var++;" ptr_var sẽ có giá trị là 1002 chứ không phải 1001



Số học con trỏ-2

<code>++ptr_var</code> or <code>ptr_var++</code>	points to next integer after var
<code>--ptr_var</code> or <code>ptr_var--</code>	points to integer previous to var
<code>ptr_var + i</code>	points to the <i>i</i> th integer after var
<code>ptr_var - i</code>	points to the <i>i</i> th integer before var
<code>++*ptr_var</code> or <code>(*ptr_var)++</code>	will increment var by 1
<code>*ptr_var++</code>	will fetch the value of the next integer after var

Mỗi lần một con trỏ được tăng lên, nó sẽ trỏ tới vị trí bộ nhớ của phần tử tiếp theo của kiểu cơ sở của nó. Mỗi lần giảm nó sẽ trỏ đến vị trí của phần tử trước đó.

Tất cả các con trỏ khác sẽ tăng hoặc giảm tùy thuộc vào độ dài của kiểu dữ liệu mà chúng trỏ tới.



So sánh con trỏ

Hai con trỏ có thể được so sánh trong một biểu thức quan hệ với điều kiện cả hai các con trỏ đang trỏ đến các biến cùng loại

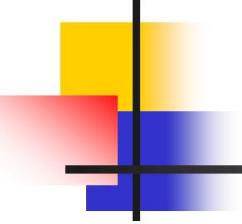
Xem xét rằng `ptr_a` và `ptr_b` là 2 biến con trỏ, trỏ đến các phần tử dữ liệu `a` và `b`. Trong trường hợp này, có thể thực hiện các so sánh sau:

<code>ptr_a < ptr_b</code>	Returns true provided a is stored before b
<code>ptr_a > ptr_b</code>	Returns true provided a is stored after b
<code>ptr_a <= ptr_b</code>	Returns true provided a is stored before b or <code>ptr_a</code> and <code>ptr_b</code> point to the same location
<code>ptr_a >= ptr_b</code>	Returns true provided a is stored after b or <code>ptr_a</code> and <code>ptr_b</code> point to the same location.
<code>ptr_a == ptr_b</code>	Returns true provided both pointers <code>ptr_a</code> and <code>ptr_b</code> points to the same data element.
<code>ptr_a != ptr_b</code>	Returns true provided both pointers <code>ptr_a</code> and <code>ptr_b</code> point to different data elements but of the same type.
<code>ptr_a == NULL</code>	Returns true if <code>ptr_a</code> is assigned NULL value (zero)

Con trỏ và Đơn Mảng chiều-1

Địa chỉ của một phần tử mảng có thể được biểu thị bằng hai cách thức:

- Bằng cách viết phần tử mảng thực tế được đặt trước dấu thăng (&)
- Bằng cách viết một biểu thức trong đó chỉ số được thêm vào tên mảng



Con trỏ và Đơn Mảng chiều-2

Ví dụ

```
#include<stdio.h> void  
main() {
```

```
    tính int ary[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}; int i; cho (i = 0; i <  
    10; i +  
    +) {
```

```
        printf("\ni=%d,aryi]=%d,*(ary+i)=%d",i, ary[i],*(ary +  
        i)); printf("&ary[i]=  
        %X,ary+i=%X",&ary[i],ary+i);  
        /* %X trả về số thập lục phân không dấu */
```

```
    }
```

```
}
```

Con trỏ và Đơn Mảng chiều-3

Đầu ra

i=0	ary[i]=1	*(ary+i)=1	&ary[i]=194	ary+i = 194
i=1	ary[i]=2	*(ary+i)=2	&ary[i]=196	ary+i = 196
i=2	ary[i]=3	*(ary+i)=3	&ary[i]=198	ary+i = 198
i=3	ary[i]=4	*(ary+i)=4	&ary[i]=19A	ary+i = 19A
i=4	ary[i]=5	*(ary+i)=5	&ary[i]=19C	ary+i = 19C
i=5	ary[i]=6	*(ary+i)=6	&ary[i]=19E	ary+i = 19E
i=6	ary[i]=7	*(ary+i)=7	&ary[i]=1A0	ary+i = 1A0
i=7	ary[i]=8	*(ary+i)=8	&ary[i]=1A2	ary+i = 1A2
i=8	ary[i]=9	*(ary+i)=9	&ary[i]=1A4	ary+i = 1A4
i=9	ary[i]=10	*(ary+i)=10	&ary[i]=1A6	ary+i = 1A6

Con trỏ và Đa Mảng chiều-1

Mảng hai chiều có thể được định nghĩa là một con trỏ
tới một nhóm các mảng một chiều liên tiếp

Chỉ dành cho Trung tâm Aptech sử dụng
Khai báo mảng hai chiều có thể được viết như sau:

```
data_type (*ptr_var) [biểu thức 2];
```

thay vì

```
data_type (*ptr_var) [biểu thức1] [biểu thức 2];
```



Con trỏ và Chuỗi-1

```
#include <stdio.h>
```

```
#include <string.h> void
```

```
main () {
```

Ví dụ

```
char a, str[81], *ptr; printf("\nNhập  
một câu:"); gets(str); printf("\nNhập ký tự để  
tìm kiếm:"); a  
= getche(); ptr = strchr(str,a); /* trả về con trỏ tới char*/  
printf( "\nChuỗi bắt  
đầu tại địa chỉ: %u",str);  
printf("\nLần xuất hiện đầu tiên của ký tự  
là tại địa chỉ: %u ",ptr); printf("\nVị trí xuất hiện đầu tiên (bắt đầu từ  
0) là: % d", ptr_str);
```

```
}
```




Con trỏ và Chuỗi-2

Đầu ra

Enter a sentence: We all live in a yellow submarine

Enter character to search for: Y

String starts at address: 65420.

First occurrence of the character is at address: 65437.

Position of first occurrence (starting from 0) is: 17



Phân bổ bộ nhớ-1

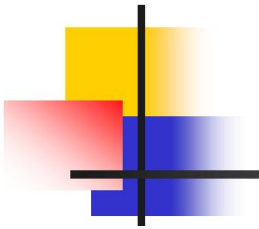
Hàm `malloc()` là một trong những hàm được sử dụng phổ biến nhất cho phép phân bổ bộ nhớ từ nhóm bộ nhớ trống. Tham số cho hàm `malloc()` là một số nguyên chỉ định số byte cần thiết.



Phân bổ bộ nhớ-2

Ví dụ

```
#include<stdio.h>
#include<malloc.h>
void main()
{
    int *p,n,i,j,temp;
    printf("\nEnter number of elements in the array :");
    scanf("%d",&n);
    p=(int*)malloc(n*sizeof(int));
    for(i=0;i<n;++i) {
        printf("\nEnter element no. %d:",i+1);
        scanf("%d",p+i); }
    for(i=0;i<n-1;++i)
        for(j=i+1;j<n;++j)
            if(*(p+i)>*(p+j)) {
                temp=*(p+i);
                *(p+i)=*(p+j);
                *(p+j)=temp; }
    for(i=0;i<n;++i)
        printf("%d\n",*(p+i));
}
```



miễn phí()-1

hàm `free()` có thể được sử dụng để giải phóng bộ nhớ khi không còn cần thiết nữa.

Cú pháp: Chỉ dành cho Trung tâm Aptech sử dụng
lệnh `void free(void *ptr);`

Hàm này giải phóng không gian được trả tới bằng cách giải con trả, phóng nó để sử dụng trong tương lai.

con trả phải được sử dụng trong một lệnh gọi trước đó tới `malloc()`, `calloc()` hoặc `realloc()`.



miễn phí()-2

```
#include <stdio.h> #include  
<stdlib.h> /*bắt buộc đối với các hàm malloc và free*/ int main() {  
  
    int number; int  
    *ptr; int i;  
    printf("Bạn  
    muốn lưu trữ bao nhiêu số nguyên? "); scanf("%d", &number); ptr = (int *) malloc  
    (number*sizeof(int)); /*phân bổ bộ  
    nhớ  
    */  
    nếu(ptr!=NULL) {  
  
        đối với (i = 0; i < số; i++) {  
  
            *(ptr+i) = i;  
  
        }  
    }  
}
```

Ví dụ

Tiếp theo.



miễn phí()-3

Ví dụ

đối với (i = số; i > 0; i--)

{

printf("%d\n",*(ptr+(i-1))); /* in ra

theo thứ tự ngược lại */

}
Chỉ dành cho Trung tâm Aptech sử dụng

free(ptr); /* giải phóng bộ nhớ được phân bổ */

trả về 0;

}

khác

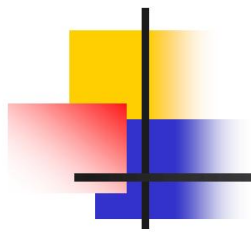
{

printf("\nKhông cấp phát được bộ nhớ - không đủ bộ nhớ.\n");

trả về 1;

}

}



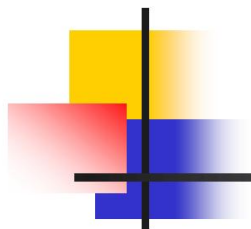
calloc() - 1

calloc tương tự như malloc, nhưng sự khác biệt chính là các giá trị được lưu trữ trong không gian bộ nhớ được phân bổ theo mặc định là số không.

calloc yêu cầu hai đối số. Đối số đầu tiên là số lượng biến bạn muốn phân bổ bộ nhớ cho. Đối số thứ hai là kích thước của mỗi biến.

Cú pháp:

```
void *calloc( size_t số, size_t kích thước );
```



calloc() - 2

Ví dụ

```
#include <stdio.h> #include  
<stdlib.h> int main() {
```

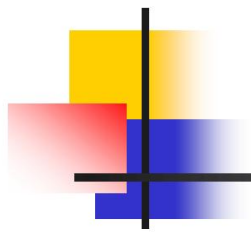
```
float *calloc1, *calloc2; int i;  
calloc1  
= (float *) calloc(3, sizeof(float)); calloc2 = (float *)calloc(3,  
sizeof(float)); nếu(calloc1!=NULL && calloc2!=NULL) {
```

```
    đối với (i = 0; i < 3; i++)
```

```
    { printf ("calloc1 [%d] giữ %05,5f ", i, calloc1 [i]); printf ("\ncalloc2 [%d]  
      giữ %05,5f ", i, * (calloc2 + i));
```

```
    }
```

Tiếp theo..



calloc() - 3

Ví dụ

```
miễn phí(calloc1);  
miễn phí(calloc2); trả  
về 0;
```

Chỉ dành cho Trung tâm Aptech sử dụng

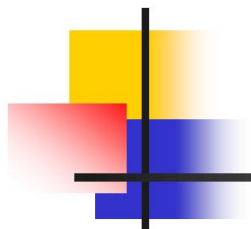
```
} khác
```

```
{
```

```
    printf("Không đủ bộ nhớ\n"); trả về 1;
```

```
}
```

```
}
```



phân bổ lại()-1

Bạn đã phân bổ một số byte nhất định cho một mảng nhưng sau đó thấy rằng bạn muốn thêm giá trị vào đó. Bạn có thể sao chép mọi thứ vào một mảng lớn hơn, điều này không hiệu quả hoặc bạn có thể phân bổ nhiều hơn byte bằng cách sử dụng `realloc` mà không làm mất dữ liệu của bạn.

Chỉ dành cho Trung tâm Aptech sử dụng

`realloc` lấy hai đối số

Đầu tiên là con trỏ tham chiếu đến bộ nhớ

Thứ hai là tổng số byte bạn muốn phân bổ lại

Cú pháp:

```
void *realloc( void *ptr, size_t kích thước );
```



phân bổ lại()-2

Ví dụ

```
#include<stdio.h>
#include <stdlib.h>
int chính()
{
    int *ptr;
    số nguyên i;
    ptr = (int *)calloc(5, sizeof(int *));
    nếu(ptr!=NULL) {

        *ptr = 1; *(ptr+1) = 2;
        ptr[2] = 4; ptr[3] = 8; ptr[4] = 16;
        ptr = (int *)realloc(ptr, 7*kích thước(int));
        nếu(ptr!=NULL)
        {
            printf("Bây giờ đang phân bổ thêm bộ nhớ... \n");
            con trỏ[5] = 32; /* Hiện nay nó hợp pháp! */
            con trỏ[6] = 64;
```



phân bổ lại()-3

Ví dụ

đối với (i = 0; i < 7; i++)

```
{  
    printf("ptr[%d] giữ %d\n", i, ptr[i]);  
}
```

```
realloc(ptr,0); /* giống như miễn phí(ptr); - chỉ là sang trọng hơn thôi! */
```

```
    trả về 0;
```

```
}
```

khác

```
{
```

```
    printf("Không đủ bộ nhớ - việc phân bổ lại không thành công.\n");
```

```
    trả về 1;
```

```
}
```

```
}
```

khác

```
{
```

```
    printf("Không đủ bộ nhớ - calloc không thành công.\n");
```

```
    trả về 1;
```

```
}
```

```
}
```