# Con trỏ (Pointer)

# Pointer

- Địa chỉ của biến
- Biến con trỏ
- Các phép toán trên biến con trỏ
- Con trỏ và mảng một chiều

# Địa chỉ của biến

- Mỗi biến của một chương trình đều được chứa trong một bộ nhớ với một địa chỉ duy nhất được cấp khi chương trình được biên dịch.
- Tùy theo kiểu dữ liệu của biến mà độ dài của biến trong bộ nhớ là khác nhau. Đơn vị được tính thao byte
- Để lấy giá trị địa chỉ của biến ta sử dụng &. Ví dụ: &x
- Để xác định kích thươc của biến thông thường, sử dụng sizeof. sizeof(x)

# Biến con trỏ

- Biến sử dụng để chứa địa chỉ của một biến khác
- Nếu một biến chứa địa chỉ của một biến khác, biến thứ nhất được gọi là trỏ tới biến thứ hai
- Biến con trỏ có thể truy xuất 1 cách gián tiếp tới dữ liệu biến thứ hai.
- Một biến con trỏ loại nào chỉ có thể lưu giữ địa chỉ (trỏ tới) các biến cùng loại.
- Để tham khảo tới nội dung biến được trỏ tới bởi biến con trỏ, sử dụng toán tử *. Ví dụ: *x
- Biến mảng là một biến con trỏ.

```
int n1=10, *n2;
//n2 là biến con trỏ có thể lưu địa chỉ của biến kiểu int
n2= &n1;
//Gán địa chỉ biến n1 cho biến con trỏ n2
printf("%d , %d, %d", n1, *n2, n2);
//Xuất nội dung biến n1, nội dung trỏ bởi n2, nội dung n2
*n2 = 20;
//Thay đổi nội dung vùng nhớ được trỏ bởi n2
printf("\n%d , %d, %d", n1, *n2, n2);
```

# Biến con trỏ

```
int a=10, b=5;
float f=1.0;
char c='A';
int M[3]={3,5,7};
printf("\nBien a, Dchi=%d, Gtri=%d, KThuoc %d bytes", &a, a, sizeof(a));
printf("\nBien b, Dchi=%d, Gtri=%d, KThuoc %d bytes", &b, b, sizeof(b));
printf("\nBien f, Dchi=%d, Gtri=%f, kich thuoc %d bytes", &f, f, sizeof(f));
printf("\nBien c, Dchi=%d, Gtri=%c, KThuoc %d bytes", &c, c, sizeof(c));
printf("\nBien M, Dchi=%d, Gtri dau=%d, KThuoc 1 phan tu %d bytes", M, M[0], sizeof(M[0]));
printf("\nBien M[0], Dchi=%d, Gtri =%d, KThuoc %d bytes", &M[0], M[0], sizeof(M[0]));
```

```
Bien a, Dchi=6487580, Gtri=10, KThuoc 4 bytes
Bien b, Dchi=6487576, Gtri=5, KThuoc 4 bytes
Bien f, Dchi=6487572, Gtri=1.000000, kich thuoc 4 bytes
Bien c, Dchi=6487571, Gtri=A, KThuoc 1 bytes
Bien M, Dchi=6487552, Gtri dau=3, KThuoc 1 phan tu 4 bytes
Bien M[0], Dchi=6487552, Gtri =3, KThuoc 4 bytes
```

# Ứng dụng biến con trỏ

- Nhận nhiều hơn một giá trị được trả vê từ hàm từ lện return.
- Truyền tham số là mảng vào hàm thuận lợi hơn
- Thao tác trên mảng dễ hơn bởi việc di chuyển con trỏ
- Cấp phát bộ nhớ động

# Các phép toán trên biến con trỏ

- Phép toán & và *
  - var2 = &var: Trả vê địa chỉ của biến var
  - Temp = *var2; truy xuất nội dung vùng nhớ trỏ bởi var2

- Mảng:
  - Là biến con trỏ
  - Tên mảng lưu giữ địa chỉ phần tử đầu tiên
  - Có thể sử dụng biến con trỏ để nhập, xuất mảng, hoặc thao tác trên mảng.

```
int M[3];
for(int i=0; i<3; i++)
        {       printf("M[%d]=", i);
                scanf("%d", M+i);
        }
for(int i=0; i<3; i++)      printf("%d ", *(M+i));
```

# Các phép toán trên biến con trỏ

- Phép toán: +, ++, --:
  - ++: Trỏ tới con trỏ kế tiếp cùng loại
  - --: Trả vê địa chỉ biến trước cùng loại
  - +i: Trả vê địa chỉ biến sau I biến

```
int n1=1, *ptr_n1, *ptr_n2, *ptr_n3;
ptr_n1=&n1; ptr_n2=&n1; ptr_n3=&n1;
printf("%d, %d", &n1, ptr_n1);//6487556, 6487556
ptr_n1++; ptr_n2--; ptr_n3=ptr_n3+2;
/* ptr_n1 trỏ tới địa chỉ số nguyên kế tiếp, ptr_n2: trỏ tới địa chỉ số nguyên trước, ptr_n3: trỏ tới địa chỉ số nguyên cách 2 ô nhớ nguyên */
printf("\n%d, %d, %d, %d", &n1, ptr_n1, ptr_n2, ptr_n3);
//6487556, 6487560, 6487552, 6487564
```

# Các phép toán trên biến con trỏ

- Phép toán so sánh:

int a, b, ptr_a=&a, ptr_b=&b;

| P | |
|---|---|
| ptr_a < ptr_b | Trả về true nếu địa chỉ biến a nhỏ hơn (lưu trữ trước) địa chỉ biến b |
| ptr_a >ptr_b | Trả về true nếu địa chỉ biến a lớn hơn (lưu trữ sau) địa chỉ biến b |
| ptr_a <=ptr_b | |
| ptr_a >=ptr_b | |
| ptr_a ==ptr_b | True nếu ptr_a và ptr_b cùng trỏ tới 1 vùng dữ liệu |
| ptr_a !=ptr_b | |
| ptr_a == NULL | True nếu ptr_a được gán giá trị NULL (zero, 0) |

# Các phép toán trên biến con trỏ

```
int n=1, *ptr_n1, *ptr_n2, M[3]={2,4,6};
ptr_n1 = &n;
ptr_n2 = ptr_n1;
*ptr_n2=10;
printf("\n n= %d, %d, %d", n, *ptr_n1, *ptr_n2); // n= 10, 10, 10
ptr_n1 = M;
ptr_n2=&M[0];
printf("\n %d, %d, %d", M[0], M[1], M[2]);// 2, 4, 6
printf("\n %d, %d, %d", M, &M[0], ptr_n1, ptr_n2);// 6487552, 6487552, 6487552
printf("\n %d, %d, %d", *M, M[0], *ptr_n1, *ptr_n2);// 2, 2, 2
ptr_n1++;
printf("\n %d , %d, %d", *(M+1), &M[1], *ptr_n1, *ptr_n2);// 4, 6487556, 4
```

# Cấp phát bộ nhớ động

- Hàm cấp phát bộ nhớ malloc
- Giải phóng bộ nhớ
- Hàm calloc

# Malloc

- Hàm malloc() được sử dụng để cấp phát một khối bộ nhớ chưa được khởi tạo. Nó trả về một con trỏ void đến byte đầu tiên của khối bộ nhớ nếu việc cấp phát thành công.
- Nếu cấp phát lỗi (bộ nhớ không đủ) kết quả trả về là NULL
- Cú pháp:

void * malloc(size);

size: Là giá trị nguyên dương đại diện cho khối bộ nhớ tính bằng byte.

```
//Ví dụ cấp phát mảng 5 số nguyên
int *ptr;
ptr = (int*) malloc(5*sizeof(int));
//5*sizeof(int) =20
if(!ptr)
   {
      printf("Khong thanh cong");
      exit(1);
   }
printf("\nThanh cong");
```

# Free

- Hàm free: giải phóng ô nhớ đã cấp phát
- Cú pháp

void free(void *ptr );

Ptr là con trỏ được cấp phát bộ nhớ bởi malloc(),calloc(), or realloc()

```
int *ptr;
ptr = (int*) malloc(5*sizeof(int));
Free(ptr);
//giải phóng ô nhớ đã cấp phát cho ptr
```

# Hàm calloc

- Hàm calloc giống malloc, sử dụng để cấp phát bộ nhớ cho biến con trỏ.
- Cú pháp:

void *calloc( size_t num, size_t size );

- Num: Số ô nhớ cần cấp phát
- Size: kích thước một ô nhớ

```
int *ptr;
ptr = (int*) calloc(5, sizeof(int));
  if(!ptr)
  {
     printf("Khong thanh cong");
     exit(1);
  }
 printf("\nThanh cong");
```

# Ví dụ

```
#include<stdio.h>
void main()
{
   int *M, n;
   printf("Nhap so phan tu n (>0)");
   scanf("%d", &n);
   M = (int*) calloc(n, sizeof(int));
   if(!M)
   {
      printf("Khong thanh cong");
      exit(1);
   }
  for(int i=0; i<n; i++)
   {
                        printf("M[%d]=", i);
                        scanf("%d", M+i);
   }
for(int i=0; i<n; i++)   printf("%d - %d ,", *(M+i), M[i]);

}
```