

Nhập môn Lập trình – IT001

Cấp phát và giải phóng bộ nhớ động

NỘI DUNG CHÍNH

- Khái niệm bộ nhớ động
- Các hàm cấp phát bộ nhớ
 - Hàm **new** (C++)
 - Hàm **malloc** (C)
 - Hàm **calloc** (C)
 - Hàm **realloc** (C)
- Hàm giải phóng bộ nhớ
 - Hàm **delete** (C++)
 - Hàm **free** (C)

Bộ nhớ động

- Bộ nhớ cấp phát cho các biến, mảng đều cố định, không thể thay đổi trong lúc chương trình chạy.
- Cần một lượng bộ nhớ mà kích cỡ của nó chỉ có thể được xác định khi chương trình chạy, ví dụ như trong trường hợp chúng ta nhận thông tin từ người dùng để xác định lượng bộ nhớ cần thiết
 - Giải pháp là dùng *bộ nhớ động*

Bộ nhớ động

- Các hàm liên quan tới bộ nhớ động:
 - new (C++)
 - delete (C++)
 - malloc (C)
 - calloc (C)
 - realloc (C)
 - free (C)

Hàm new (C++)

- **Chức năng:** cấp phát bộ nhớ. Hàm này trả về một con trỏ trỏ tới đầu của khối nhớ vừa được cấp phát.

- **Cú pháp:**

Cấp phát bộ nhớ chứa một phần tử có kiểu *type*:

pointer = **new** *type*

Cấp phát một khối nhớ (một mảng) gồm các phần tử kiểu *type*:

pointer = **new** *type* [*elements*]

- **Ví dụ:**

int *p; **p= new int;**

int *c; **c= new int[10];**

Hàm new (C++)

```
int *p;    p= new int;
```



```
int *c;    c= new int[3];
```



Hàm new (C++)

- Việc cấp phát bộ nhớ cho 1 con trỏ có khác gì so với khai báo mảng thông thường?
- Bộ nhớ động được quản lí bởi hệ điều hành và trong các môi trường đa nhiệm có thể chạy một lúc vài chương trình, có thể hết bộ nhớ để cấp phát. Nếu hệ điều hành không thể cấp phát bộ nhớ khi dùng với toán tử **new**, một con trỏ **NULL** sẽ được trả về.
- Nên kiểm tra xem con trỏ trả về bởi toán tử **new** có bằng NULL hay không:

```
int * a;
```

```
a = new int [5];
```

```
if (a == NULL) { // Thông báo hết bộ nhớ. };
```

Hàm delete (C++)

- Vì bộ nhớ động chỉ cần thiết trong một khoảng thời gian nhất định, khi không cần dùng đến nữa thì giải phóng nó để cấp phát cho các nhu cầu khác trong tương lai.
- Để thực hiện việc này dùng toán tử **delete**:
Giải phóng bộ nhớ được cấp phát cho một phần tử
delete pointer ;
Hoặc giải phóng một khối nhớ gồm nhiều phần tử (mảng)
delete [] pointer ;
- Trong hầu hết các trình dịch cả hai biểu thức là tương đương mặc dù chúng là rõ ràng là hai toán tử khác nhau.

Hàm delete (C++)

```
int *p; p = new int; delete p;
```



- Xóa ô nhớ 1000 rồi nhưng p vẫn trỏ tới ô nhớ có địa chỉ 1000!
 - Gọi là “con trỏ lạc”
- Nếu sau đó lỡ thao tác trên p (*p)
 - Kết quả không lường trước được!
 - Thường là nguy hiểm!
- Hãy tránh con trỏ lạc: gán con trỏ bằng NULL sau khi delete


```
delete p; p = NULL;
```

Hàm malloc (C)

- **Chức năng:** cấp phát bộ nhớ. Hàm này trả về một con trỏ kiểu void*.
- **Nguyên mẫu:**

void * malloc (size_t *nbytes*);

Trong đó: *nbytes* là số byte chúng ta muốn gán cho con trỏ

- **Sử dụng:** vì hàm này trả về một con trỏ kiểu void*, nên ta phải chuyển đổi kiểu sang kiểu của con trỏ đích

Ví dụ:

```
char * ronny;  ronny = (char *) malloc (sizeof(char));
```

```
char * tom;    tom= (char *) malloc (sizeof(char)*5);
```

Toán tử *sizeof* trả về kích thước của một kiểu dữ liệu cụ thể.

Hàm calloc (C)

- **Chức năng:** cấp phát bộ nhớ. Hàm này trả về một con trỏ kiểu void*.
- **Nguyên mẫu:**

void * calloc (size_t *nelements* , size_t *size*);

Trong đó: *nelements*: số phần tử

size: kích thước của mỗi phần tử.

Hàm này cấp phát bộ nhớ có kích thước bằng *nelements** *size*

- **Sử dụng:** giống hàm malloc

Ví dụ:

```
char * ronny;  ronny = (char *) calloc (1, sizeof(char));
```

```
char * tom;    tom= (char *) calloc (5, sizeof(char));
```

Hàm realloc (C)

- **Chức năng:** thay đổi kích thước của khối nhớ đã được cấp phát cho một con trỏ.
- **Nguyên mẫu:**

void * realloc (void * *pointer* , size_t *newsize*);

Trong đó: *pointer*: con trỏ đã được cấp phát bộ nhớ (với các hàm new, malloc, calloc, realloc trước đó) hay một con trỏ null

newsize: kích thước của khối nhớ mới

Hàm realloc (C)

- Nếu kích thước khối nhớ mới *newsize* nhỏ hơn kích thước cũ: *pointer* vẫn giữ nguyên địa chỉ, giá trị các phần tử dư thừa bị xóa đi. Hàm trả về địa chỉ của *pointer*.
- Nếu kích thước khối nhớ mới *newsize* bằng kích thước cũ: *pointer* vẫn giữ nguyên địa chỉ và giá trị các phần tử. Hàm trả về địa chỉ của *pointer*.
- Nếu kích thước khối nhớ mới *newsize* lớn hơn kích thước cũ: cấp phát 1 khối nhớ mới và các phần tử có giá trị bằng giá trị các phần tử trong *pointer*; *pointer* vẫn giữ nguyên địa chỉ và giá trị tất cả phần tử bị xóa đi. Hàm trả về địa chỉ của khối nhớ mới.

Hàm realloc (C)

- Nếu không thể thay đổi kích thước của *pointer* thì hàm sẽ trả về một con trỏ NULL, *pointer* và nội dung của nó sẽ không bị thay đổi

- **Sử dụng:**

```
int *p = (int*)malloc(5*sizeof(int));  
int *q = (int*)realloc(p, 10*sizeof(int));
```

Hàm free (C)

- **Chức năng:** giống delete
- **Nguyên mẫu:**

`void free (void * pointer);`

Trong đó: *pointer* là con trỏ cần giải phóng bộ nhớ

- **Sử dụng:** giống hàm malloc

Ví dụ:

```
int* my_array = (int*)malloc(10*sizeof(int));  
free(my_array);
```

Hàm free (C) – Mảng 1 chiều

```
a = (int *)malloc(n*sizeof(int));  
/* kiểm tra sự cấp phát thành công */  
if(a!=NULL)  
{  
.....  
    free(a);  
}
```


Hàm free (C)-Mảng 2 chiều

```
a = (int **)malloc(m*sizeof(int *));
```

```
/* kiểm tra sự cấp phát thành công */  
if(a!=NULL)  
{ kt=0;  
  for(i=0;i<m;i++)  
    a[i]=NULL;  
  for(i=0;i<m;i++)  
    { if(kt==1) break;  
      a[i]=(int *)malloc(n*sizeof(int));  
      if(a[i]==NULL) kt=1;  
    }
```

```
if(kt==0)  
  { /* sử dụng được a[i][j]  
    */  
    .....  
    for(i=0;i<m;i++)  
      if(a[i]!=NULL)  
        free(a[i];  
        free(a);  
    }
```

Bài tập 1

```
1  //Program to demonstrate pointers and dynamic variables.
2  #include <iostream>
3  using std::cout;
4  using std::endl;

5  int main()
6  {
7      int *p1, *p2;

8      p1 = new int;
9      *p1 = 42;
10     p2 = p1;
11     cout << "*p1 == " << *p1 << endl;
12     cout << "*p2 == " << *p2 << endl;

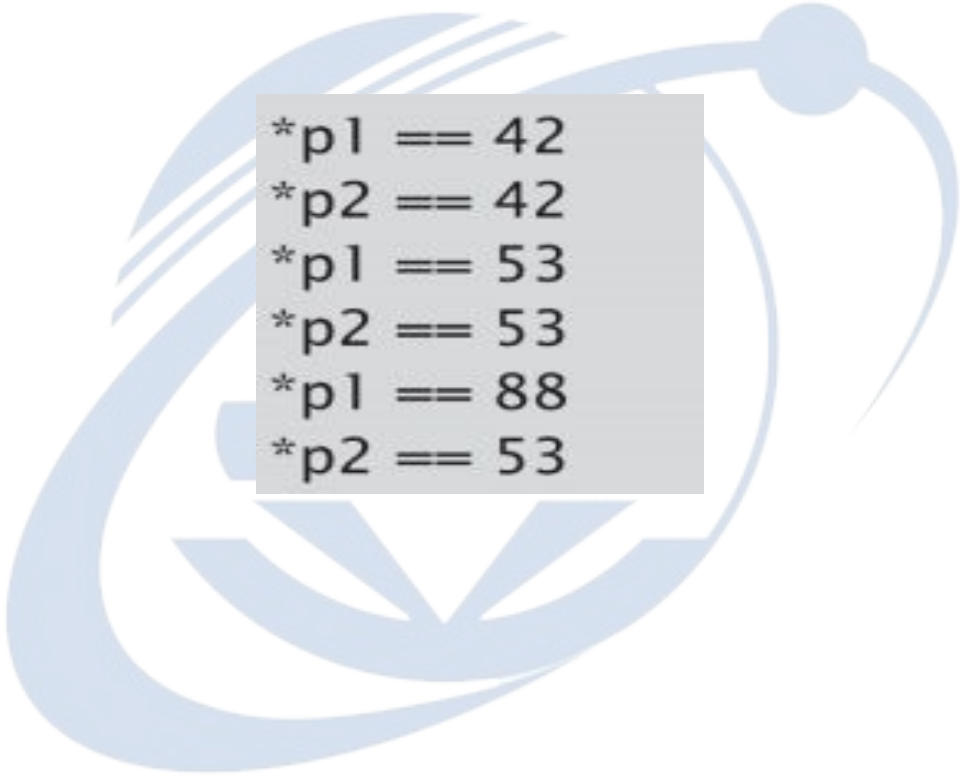
13     *p2 = 53;
14     cout << "*p1 == " << *p1 << endl;
15     cout << "*p2 == " << *p2 << endl;
```

Bài tập 1

```
16     p1 = new int;  
17     *p1 = 88;  
18     cout << "*p1 == " << *p1 << endl;  
19     cout << "*p2 == " << *p2 << endl;  
  
20     cout << "Hope you got the point of this example!\n";  
21     return 0;  
22 }
```



Bài tập 1



```
*p1 == 42  
*p2 == 42  
*p1 == 53  
*p2 == 53  
*p1 == 88  
*p2 == 53
```