Dynamic Memory

Bùi Tiến Lên

2024

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Contents

1. Memory Management

2. Dynamic Memory in C

3. Dynamic Memory in C++

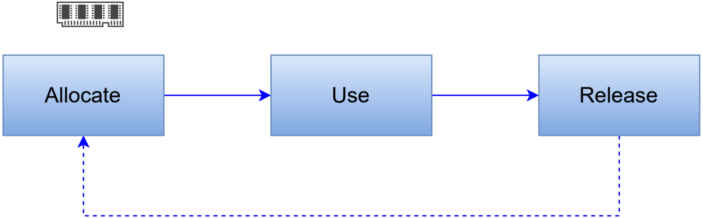
4. Workshop

Memory Management

Introduction

In C/C++,

- when we create variables, objects, or anything you can think of, the machine allocates memory for this
- when we don't need them, release them.



Introductiont (cont.)

Memory management is central to all programs.

1. Memory is managed by the runtime system implicitly, such as when memory is allocated for automatic variables.
2. Memory is managed explicitly. The ability to allocate and then deallocate memory allows an application to manage its memory more efficiently and with greater flexibility.

The memory heap and stack

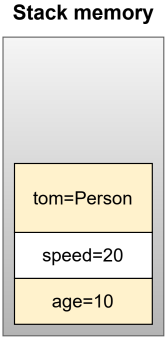
C/C++ uses two common places to store objects

- Stack memory
- Heap memory

Stack: Static memory allocation

- Stack memory is used to store static data such as local variables and parameters where C/C++ knows at compile time.

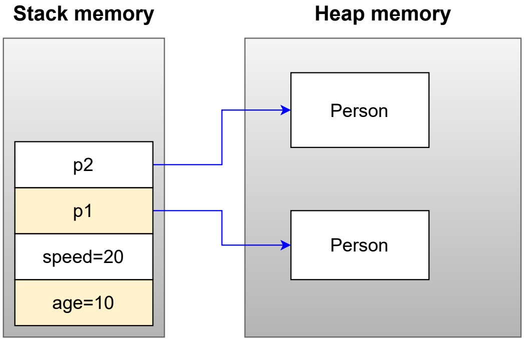
```
int age=10;
int speed=20;
Person tom = Person;
```



Heap: Dynamic memory allocation

- Heap memory is used to store dynamic data such as dynamic objects.

```
int age=10;
int speed=20;
Person *p1 = new Person;
Person *p2 = new Person;
```



Dynamic Memory in C

Dynamic Memory Allocation Functions

- The library `cstdlib` or `stdlib.h`

Function	Description
`malloc`	Allocates memory from the heap
`realloc`	Reallocates memory to a larger or smaller amount based on a previously
`calloc`	Allocates and zeros out memory from the heap
`free`	Returns a block of memory to the heap

malloc

- The `malloc(...)` function is the simplest of the allocation functions. It takes a single argument, which is the size of the block you want to allocate, and it returns a newly allocated memory block of that size. The return type is `void *`.

```
int *ip = (int *)malloc(sizeof(int));
int *ips = (int *)malloc(10 * sizeof(int));
double *dp = (double *)malloc(sizeof(double));
struct Point {
  double x;
  double y;
};
Point *p = (Point *)malloc(sizeof(Point));
```

malloc

- If `malloc(...)` cannot allocate the memory you want, for example, if there isn't sufficient memory in the process' memory space (or for whatever other reasons), it will return a NULL pointer.

```
char *p = (char *)malloc(1000000000);
if (p != nullptr) {
  cout << "Success\n";
} else {
  cout << "Fail\n";
}
```

calloc

- The `calloc(...)` function takes two arguments, the size of the elements you want to allocate plus how many elements you want.

```
int *ips = (int *)calloc(10, sizeof(int));
double *dps = (double *)calloc(20, sizeof(double));
```

realloc

- If you need to resize a chunk of memory that you have allocated using one of the other functions, then you go to realloc(...)

```
int *ips = (int *)malloc(10 * sizeof(int));
...
int *new_ips = (int *)realloc(ips, 100 * sizeof(int));
```

- What happen?

```
int *ips = (int *)malloc(10 * sizeof(int));
...
ips = (int *)realloc(ips, 100 * sizeof(int));
```

free

- If you allocate memory on the heap, you must free it.

```
int *pi = (int*) malloc(sizeof(int));
...
free(pi);
```

Memory Leaks

Concept 1

A memory leak occurs when allocated memory is never used again but is not freed.

Memory leaks can happen when:

- The memory's address is lost
- The `free` function is never invoked though it should be (sometimes called a hidden leak)

Losing the address

```
int *pi = (int*) malloc(sizeof(int));
*pi = 5;
...
pi = (int*) malloc(sizeof(int));
```

Hidden memory leaks

```
struct Person {
    char *firstName;
    char *lastName;
    char *title;
    unsigned int age;
};
Person *p;
...
delete p;
```

Dynamic Memory in C++

Operators new

- You use `new` to allocate new memory blocks. The most frequently used form of `new` returns a pointer to the requested memory if successful or else throws an exception. When using `new`, you need to specify the data type for which the memory is being allocated:

  ```
  Type* Pointer = new Type; // request memory for one element
  ```

- You can also specify the number of elements you want to allocate that memory for (when you need to allocate memory for more than one element):

  ```
  Type* Pointer = new Type[NumElements]; // request memory for NumElements
  ```

Operators new (cont.)

```
int* pNumber = new int;  // get a pointer to an integer
int* pNumbers = new int[10];  // get a pointer to a block of
    10 integers
```

Operator delete

- Every allocation using `new` needs to be eventually released using an equal and opposite de-allocation via `delete`:

  ```
  Type* Pointer = new Type;
  delete Pointer; // release memory allocated above for one instance of Type
  ```

- This rule also applies when you request memory for multiple elements:

  ```
  Type* Pointer = new Type[NumElements];
  delete[] Pointer; // release block allocated above
  ```

Comparison of new/delete vs malloc/free

Feature	new / delete	malloc / free
Memory allocated from	'Free Store'	'Heap'
Returns	Fully typed pointer	void*
On failure	Throws (never returns NULL)	Returns NULL
Required size	Calculated by compiler	Must be specified in bytes
Handling arrays	Has an explicit version	Requires manual calculations
Reallocating	Not handled intuitively	Simple

Workshop

Quiz

1. What is dynamic memory management?

   ....

   ....

   ....

Exercises

- Write a program

References

- Deitel, P. (2016).
  C++: How to program.
  Pearson.

- Gaddis, T. (2014).
  Starting Out with C++ from Control Structures to Objects.
  Addison-Wesley Professional, 8th edition.

- Jones, B. (2014).
  Sams teach yourself C++ in one hour a day.
  Sams.