Module 10: Programming in C++

Dynamic Memory Management

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Tanwi Mallick
Srijoni Majumdar
Himadri B G S Bhuyan

Module Objectives

- Understand the dynamic memory management in C++

Module Outline

- Memory management in C
  - `malloc()` & `free()`
- Memory management in C++
  - `new` and `delete`
  - Array `new[]` and `delete[]`
  - Placement `new()`
  - Restrictions
- Overloading `new` and `delete`

Program 10.01/02: malloc() & free(): C & C++

C Program

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


int main() {
    int *p = (int *)malloc(sizeof(int));
    *p = 5;

    printf("%d", *p);

    free(p);

    return 0;
}
-----
5
```

C++ Program

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    int *p = (int *)malloc(sizeof(int));
    *p = 5;

    cout << *p;

    free(p);

    return 0;
}
-----
5
```

- Dynamic memory management functions in `stdlib.h` header for C (`cstdlib` header for C++)
- `malloc()` allocates the memory on heap
- `sizeof(int)` needs to be provided
- Pointer to allocated memory returned as `void *` – needs cast to `int *`
- Allocated memory is released by `free()` from heap
- `calloc()` and `realloc()` also available in both languages

Program 10.02/03: operator new & delete: Dynamic memory management in C++

- C++ introduces operators `new` and `delete` to dynamically allocate and de-allocate memory:

`malloc()` & `free()`

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    int *p = (int *)malloc(sizeof(int));
    *p = 5;

    cout << *p;

    free(p);

    return 0;
}
-----
5
```

- Function `malloc()` for allocation on heap
- `sizeof(int)` needs to be provided
- Allocated memory returned as `void *`
- Casting to `int *` needed
- Cannot be initialized
- Function `free()` for de-allocation from heap
- Library feature – header `cstdlib` needed

Operators `new` & `delete`

```
#include <iostream>

using namespace std;

int main() {
    int *p = new int(5);


    cout << *p;

    delete p;

    return 0;
}
-----
5
```

- Operator `new` for allocation on heap
- No size specification needed, type suffices
- Allocated memory returned as `int *`
- No casting needed
- Can be initialized
- Operator `delete` for de-allocation from heap
- Core language feature – no header needed

Program 10.02/04: Functions: operator new() & operator delete()

- C++ also allows `operator new` and `operator delete` functions to dynamically allocate and de-allocate memory:

`malloc()` & `free()`

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    int *p = (int *)malloc(sizeof(int));
    *p = 5;

    cout << *p;

    free(p);

    return 0;
}
-----
5
```

- Function `malloc()` for allocation on heap
- Function `free()` for de-allocation from heap

`new` & `delete`

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main(){
    int *p = (int *)operator new(sizeof(int));
    *p = 5;

    cout << *p;

    operator delete(p);

    return 0;
}
-----
5
```

- Function `operator new()` for allocation on heap
- Function `operator delete()` for de-allocation from heap

There is a major difference between operator `new` and function `operator new()`. We explore this angle more after we learn about classes

Program 10.05/06: Operators new[] & delete[]: Dynamically managed Arrays in C++

malloc() & free()

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main() {
    int *a = (int *)malloc(sizeof(int)* 3);
    a[0] = 10; a[1] = 20; a[2] = 30;

    for (int i = 0; i < 3; ++i)
        cout << "a[" << i << "] = "
             << a[i] << "     ";
    cout << endl;

    free(a);

    return 0;
}
-----
a[0] = 10     a[1] = 20     a[2] = 30
```

- Allocation by `malloc()` on heap
- # of elements implicit in size passed to `malloc()`
- Release by `free()` from heap

new[] & delete[]

```
#include <iostream>
using namespace std;


int main() {
    int *a = new int[3];
    a[0] = 10; a[1] = 20; a[2] = 30;

    for (int i = 0; i < 3; ++i)
        cout << "a[" << i << "] = "
             << a[i] << "     ";
    cout << endl;

    delete [] a;

    return 0;
}
-----
a[0] = 10     a[1] = 20     a[2] = 30
```

- Allocation by `operator new[]` (**different from** `operator new`) on heap
- # of elements explicitly passed to `operator new[]`
- Release by `operator delete[]` (**different from** `operator delete`) from heap

Program 10.07: Operator new(): Placement new in C++


```
#include <iostream> using namespace std;
int main() {
    unsigned char buf[sizeof(int)* 2]; // Buffer on stack

    // placement new in buffer buf
    int *pInt = new (buf) int (3); int *qInt = new (buf+sizeof(int)) int (5);

    int *pBuf = (int *)(buf + 0); int *qBuf = (int *)(buf + sizeof(int));
    cout << "Buf Addr  Int Addr" << endl;
    cout << pBuf << "  " << pInt << endl << qBuf << "  " << qInt << endl;
    cout << "1st Int  2nd Int" << endl;
    cout << *pBuf << "        " << *qBuf << endl;

    int *rInt = new int(7); // heap allocation
    cout << "Heap Addr  3rd Int" << endl;
    cout << rInt << "   " << *rInt << endl;
    delete rInt;            // delete integer from heap

    // No delete for placement new

    return 0;
}
-----
Buf Addr  Int Addr
001BFC50  001BFC50
001BFC54  001BFC54
1st Int  2nd Int
3        5
Heap Addr  3rd Int
003799B8   7
```

- Placement new operator takes a buffer address to place objects
- These are not dynamically allocated on heap – may be allocated on stack
- Allocations by Placement new operator must not be deleted

Mixing malloc, operator new, etc

- Allocation and De-Allocation must correctly match. Do not free the space created by new using `free()`. And do not use `delete` if memory is allocated through `malloc()`. These may results in memory corruption

Allocator	De-allocator
`malloc()`	`free()`
`operator new`	`operator delete`
`operator new[]`	`operator delete[]`
`operator new()`	No delete

- Passing `NULL` pointer to `delete` operator is secure
- Prefer to use only `new` and `delete` in a C++ program
- The `new` operator allocates exact amount of memory from Heap
- `new` returns the given pointer type – no need to typecast
- `new`, `new[ ]` and `delete`, `delete[]` have separate semantics

Program 10.08: Overloading operator new


```
#include <iostream>
#include <stdlib.h>
using namespace std;

void* operator new(size_t n) { // Definition of new
    cout << "Overloaded new" << endl;
    void *ptr;
    ptr = malloc(n);           // Memory allocated to ptr
    return ptr;
}
void operator delete(void *p) { // definition of delete
    cout << "Overloaded delete" << endl;
    free(p);                    // Allocated memory released
}
int main() {
    int *p = new int; // calling overloaded operator new
    *p = 30;          // Assign value to the location
    cout << "The value is :\t" << *p << endl;
    delete p;         // calling overloaded operator delete
    return 0;
}
-----
Overloaded new
The value is :  30
Overloaded delete
```

- `operator new` overloaded
- The first parameter of overloaded `operator new` must be `size_t`
- The return type of overloaded `operator new` must be `void *`
- The first parameter of overloaded `operator delete` must be `void *`
- The return type of overloaded `operator delete` must be `void`
- More parameters may be used for overloading
- `operator delete` should not be overloaded (usually) with extra parameters

Program 10.09: Overloading operator new[]


```
#include <iostream>
#include <cstdlib>
using namespace std;

void* operator new [] (size_t os, char setv) { // Fill the allocated array with setv
    void *t = operator new(os);
    memset(t, setv, os);
    return t;
}

void operator delete[] (void *ss) {
    operator delete(ss);
}

int main() {
    char *t = new('#')char[10]; // Allocate array of 10 elements and fill with '#'

    cout << "p = " << (int) (t) << endl;
    for (int k = 0; k < 10; ++k)
        cout << t[k];

    delete [] t;
    return 0;
}
-----
p = 19421992
##########
```

- operator new[] overloaded with initialization
- The first parameter of overloaded operator new[] must be size_t
- The return type of overloaded operator new[] must be void *
- Multiple parameters may be used for overloading
- operator delete [] should not be overloaded (usually) with extra parameters

Module Summary

- Introduced `new` and `delete` for dynamic memory management in C++
- Understood the difference between `new`, `new[]` and `delete`, `delete[]`
- Compared memory management in C with C++
- Explored the overloading of `new`, `new[]` and `delete`, `delete[]` operators

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, *Instructor*	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, *TA*	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, *TA*	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, *TA*	himadribhuyan@gmail.com	9438911655