Introduction to the Free Store

Overview

- C and Dynamic Memory
- C++ and Creating Single Objects
- C++ and Arrays

2

C and Malloc

- In ANSI C you use a malloc function for dynamic memory
- Creation of a single value or array is the same, no syntax differences

```
void main()
{
  int* p;
  p = (int*)malloc(1 * sizeof(int));
  free(p);

  p = (int*)malloc(10 * sizeof(int));
  free(p);
}
```

3

C AND MALLOC

When you use arrays in C you have two options. A static or dynamic array. A static array is the most straightforward since the declaration shows that it is an array because of the brackets. When you want a bit of more flexibility you have to use dynamic arrays. Dynamic arrays are created on the heap or free store. Both names are used to identify the memory that you can manage outside of the memory your program code uses. This memory needs to be managed by you. When you ask the system for a piece of memory it tries to allocate it and if it succeeds it returns a handle to the address of that memory. The functions provided in C do not know what the memory is used for. You call the functions supplying a number of bytes to allocate which it does. When there is no memory it returns NULL. This memory can be used for all kind of purposes one of them being an array. If you look at the example on the sheet you can see that the first call to malloc allocates a piece of memory the size of one integer value. This memory is large enough for one value. The second example allocates a piece of memory large enough to hold 10 integer values. It is an array of 10 integers. In the example you can see that the malloc function does not know if it is an array or a single value that is going to be allocated. The function only needs to know how much memory it needs to reserve.

C++ and The Free Store

- Allocate memory resources at run-time
- Two operators
  - for allocating `new`
  - for deallocating `delete`

4

FREE STORE

It is possible to allocate memory during run-time by the use of so called dynamic memory. In C we allocate (request) memory by using a number of functions such as malloc() and calloc().

These functions attempt to allocate the memory from the free store and return a pointer to the allocated memory. The C++ equivalent of these functions is the operator new(). This operator is not a function but an operator which therefore can be overloaded.

The new() operator also tries to request the memory from the heap and returns a pointer to the allocated memory. This pointer must also be checked to see if the memory is available.

When deallocating memory (freeing) you would use the free() function in C, in C++ it is also an operator, delete.

When creating a C++ program it is still possible to use the C functions for allocating and deallocating memory, but what you need to remember is not to mix these types of memory usage. When requesting memory with malloc() do not free it using delete or vice versa. Unpredictable things can happen.

Using the free store can be divided in

- pointers to objects
- arrays of objects

Pointers to Objects
(atomic types)

- Allocation of single variable

```
int* pi;
pi = new int;
```

- Deallocation

```
delete pi;
```

5

POINTERS TO OBJECTS

The simplest case possible is to allocate memory to store a value of an atomic type like int, char, float and double. The new operator returns a pointer to a piece of memory that is large enough to hold a value of that specific type. The allocated memory needs to be freed using the delete operator.

SYNTAX OF *NEW*

The syntax of the new operator is quite simple use the keyword new followed by the type for which you want to reserve a piece of memory.

```
new <type>;
```

The new Operator

```
void main()
{
        int* pi;
        float* fp;
        pi = new int;
        pf = new float;
}
```

Usage of new

The new operator allocates a piece of memory large enough to hold a value of that specific atomic type. If no memory is available the operator returns NULL.

SYNTAX OF *DELETE*

The memory can be deallocated (freed) by using the delete operator.

The syntax for delete when used for pointers to single values;

```
delete <pointer_to_memory>;
```

The delete Operator

```
void main()
{
        int* pi;
        float* fp;
        pi = new int;
        pf = new float;

        delete pi;
        delete pf;

}
```

Usage of delete

Pointers to Objects
(user defined types)

- Create objects dynamically

```
Point* pp;
pi = new Point;  // Default constructor is called
```

- Deallocation

```
delete pp;       // Destructor gets called
```

6

ALLOCATING OBJECTS (USER-DEFINED TYPES)

The new operator can be used to allocate memory for holding objects as well. The syntax is the same.

New and Objects

```
void main()
{
   Date* pd;
   Point* pp;

   pd = new Date;
   pp = new Point;
}
```

For the type you can use a class name for example Point or Date. The new operator allocates a piece of memory to store an object of the specified type and calls for that block of memory, which is in fact an object, the default constructor.

Using the delete operator for objects is the same as for the atomic types. The destructor is called for the dynamically created object.

delete and Objects

```
void main()
{
   Date* pd;
   Point* pp;

   pd = new Date;
   pp = new Point;

   delete pd;
   delete pp;

}
```

Other Constructors

- For objects it is possible to call other constructors

```
Point* pp;

pp = new Point;              // Default constructor
delete pp;

pp = new Point(10, 30);      // Other constructor
delete pp;
```

7

Calling Other Constructors

The second possibility is to call another constructor than the default constructor for the allocated object.

Syntax

```
new <type>(constructor_arguments);
```

This syntax of the new operator allocates the object on the heap, calls it specified constructor and returns a pointer to the object.

Usage

```
void main()
{
   Date* dp;
   Date* dp2;

   dp = new Date;                // Default constructor called
   dp2 = new Date(10,10,1995);   // Constructor with day, month and year
   ...
   ...
   delete dp;
   delete dp2;
}
```

Using Pointers to Objects

- Call member functions using the '->'

```
Point* pp;
pp = new Point;
pp->SetX(10.0);
```

- Dereference

```
(*pp).SetX(10.0);
```

Arrays and the Free Store

- Creating arrays

```
Point* pp;
pp = new Point[10];
```

- Deleting arrays

```
delete[] pp;
```

- Cannot call other than default constructor

9

ARRAYS AND THE FREE STORE

An other way of using the new operator is by allocating arrays of object instead of single objects.

This type of allocation is slightly different from the previous one.

```
new <type>[<size>];

delete [] <pointer_to_memory>;
```

Syntax

The new operator allocates a contiguous block of memory large enough to hold <size> number of objects of type <type>. For each object on the array, the default constructor gets called. It is NOT possible to specify a different constructor to be called on creation. The default constructor is called always.

Freeing memory is executed by using the delete operator. You have to specify that the pointer is a pointer to a block of memory which was allocated as an array.

The destructor is called for each object in the array. Do not forget the []. What happens if you forget these brackets is unpredictable.

A few rules when using the free store:

- Do not mix the C and C++ free store.
- When allocating a single object use the delete without brackets.
- When allocating an array use the delete with brackets.
- Check what new returns to see if its NULL.

Tips and Pitfalls for Free Store

- Watch out for the [] with delete
- Do not mix C allocation with C++ allocation and vice versa
- Do not forget to delete allocated memory