SFU CMPT-212 2008-1 Topic: Memory Allocation

Ján Maňuch

E-mail: jmanuch@sfu.ca

Monday 28th January, 2008

Methods of allocating memory

- *automatic storage* — variables of a function or defined inside of a block (`{ ... }`) — exist only during the execution of the function/block
- *static storage* — external (outside of the functions) and static (declared with the keyword `static`) definitions of variables — exist throughout the execution of the entire program
- *dynamic storage* — lifetime of the data is controlled by a programmer — data starts to exist after calling the `new` operator, and seizes to exist after calling the `delete` operator

Dynamic storage

- allocating memory during the runtime:
  `new type` — returns a pointer to a new allocated memory with `sizeof(type)` bytes

  Examples:

```
1   int *p=new int;
2   Point *q;
3   q=new Point;
```

  Example: usenew.cpp [Prata]

- freeing the allocated memory (so that it can be reused):
  `delete pointer` — pointer should point to the memory allocated by `new` operator

  Examples:

```
1   delete p;
2   delete q;
```

- Rules to remember:
  - you should call `delete` for each address pointing to memory allocated with the `new` operator — otherwise *memory leaks*
  - call it only once for each block of memory

Note: you don't have to use the same pointer to free the memory.

Example:

```
1  double *p=new double;
2  double *q;
3
4  q=p;
5
6  delete q;
```

Dynamic arrays

- creating a dynamic array:
  `new type[size]` — returns a pointer to the new allocated memory block containing an array of type `type` and size `size` (`size*sizeof(type)` bytes)
- freeing a dynamic array:
  `delete [] pointer` — where `pointer` should point to the memory block containing a dynamic array

Example:

```
1  int func(int n)
2  {
3    double *da=new double[n];
4    operations using da[0],..,da[n-1]
5    delete [] da;
6    return computed value
```

```
7       }
```

Important: never mix `delete []` and `delete`.

- if you allocate memory with `new`, delete it with `delete`
- if you allocate memory (an array) with `new []`, delete it with `delete []`

Avoiding crashes

When working with pointers and dynamic memory it's easy to write an incorrect code which results in a program crash. Keep in mind:

- Declaration of pointer does not allocate memory!

```
1 double *p;
2 *p=5.6; // crash
```

- Avoid memory leaks: for each `new` in your program, you should be able to quickly say where the corresponding `delete` is.
- Make sure you are not calling `delete` twice on the same block of memory. (Note that `delete NULL;` will do nothing.)
- Remember to call `delele []` instead of `delete` for dynamic arrays.
- When using pointers to build complex structures like linked lists, always draw pictures of how the memory looks like.

Sample questions

- The data in C++ can have three types of *storage*. List all three types. For each type, explain:
  1. how to create a data having the particular type of storage,
  2. when the memory for the data is allocated, and
  3. when the memory for the data is freed.
- What's wrong with the following code?

```
1  int *p;
2  {
3    int a=10;
4    p = &a;
5  }
6  cout <<*p;
```

- What is wrong with the following code?

```
1 double *x=new double[30];
2 ...
3 delete x;
```

- Implement stack of `int`s which has no limit on the number of elements stored in it.
- Implement the following function

```
1 int* increaseCapacity(int* array,int oldsize,
2                        int newsize)
```

  which increases the capacity of an array.
- Explain memory leak.