Lecture 21

CIS 208

Wednesday, April 20, 2005
Constructors

- Initialization functions for classes
- Automatically called when object is created.
- May be overloaded.
Constructors

- No return types.
- Name of constructor must be same as class.
- implemented just like any other function.
Constructors

- global variables: called before main()
- local: called when declared.
- Not necessary.
class X {
    int a;
public:
    X(int j) { a = j; }
    int geta() { return a; }
};

int main() {
    X ob = 99;
    X ob(99);
    cout << ob.geta();
    return 0;
}
single parameter constructors

- X ob = 99; == X ob(99);

- only works on constructors with one parameter.
Deconstructors

- complement of constructors

- Automatically called when object is destroyed.

- global: after program termination
- local: after code segment
Deconstructors

- class foo;

- constructor == foo(...);
- deconstructor == ~foo();

- no parameters.
why use destructors?

- Deal with last minute issues.
- Unallocated any memory.
- Close a file.
Passing Objects

- Usual call-by-value vs. call-by-reference

- CBV: separate copy of object made
  - Exact copy, all value identical
  - regular constructor is not called

- Deconstructor is called.
pointers to objects

- Work just like C pointers.

- use -> to access class functions/fields.
pointers

class cl{
  int j;
public :
cl(int k) { j = k; }
int getJ() { return j; }
};

int main()
{
  cl ob(88), *p;
  p = &ob;
  cout << p -> getJ();
  return 0;
}
passing pointers

- Call by reference.

- No constructor or deconstructor when creating pointers.
dynamic memory allocation

- 2 new commands.
  - new
  - delete

- a lot like malloc() & free()
new

- returns a pointer to the new allocated memory space

- the pointer is of the correct type.
  - casting isn’t necessary
delete

- destroys memory pointed to by pointer.

- Just like free()

- unlike free(), delete calls deconstructor, and you can try to delete unallocated pointers.
int main() {
    int *p,*q;

    p = new int;
    *p = 100;
    q = new int(5);
    cout << "At " << p << " ";
    cout << "is the value " << *p << " \n";
    delete p;
    return ;
initializing allocation memory

- `p_var = new var_type(initial data);`

- `cl *p = new cl(99);`
  - creates new `cl` object, sends 99 to constructor.


```cpp
delete p;
```
if using NULL, need to #include <cstdlib>
recursive constructors

class foo{
  foo *child;
  public :
  foo();
  ~foo();
};
continued

    foo:: foo() {
        child = new foo;
    }

    foo::~foo() {
        delete child;
    }
bad allocation

- new doesn’t return NULL if it fails.
- It actually throws an exception.
- We’ll talk about exception handling Friday.
making new return NULL;

- use nothrow operator.

\[ p\_var = \text{new}(\text{nothrow}) \text{ type}; \]

- return NULL on failure

- must #include \textless new\textgreater for this.