CMPS 109 2-23-11

- mid 2: mon 3/7
- asg 4: due wed 3/2
  submit clin is open.

Templates

Ex
```
void swap(int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

make data type (int) into a "variable".

Ex.
```
template<typename T>
void swap(T& x, T& y)
{
    T temp = x;
    x = y;
    y = temp;
}
```

in fcn main()
```
int a=6, b=7; string s1="", s2="";
swap(a, b);
swap(s1, s2);
```

Note: "typename" is equiv. to "class" in this context.

Can also do:

```
template<typename T1, typename T2>
  :
```

See: swap.cpp

recall:

```
int a=6, b=7;
double c=8.1, d=9.2;
swap (a, b);  swap(c,d);
swap<int>(a,b); swap<double>(c,d);
swap<int>(c,d);
```

```
// swap.cpp
#include<iostream>
using std::cout;
using std::endl;
using std::string;

// swap(): interchanges two values of any type
template<typename T>
void swap(T& x, T& y)
{
   T temp = x;
   x = y;
   y = temp;
}

int main()
{
   int a = 2, b = 3;
   double c = 4.5, d = 6.7;
   char e = 'q', f = 'r';
   string s1 = "happy";
   string s2 = "sad";

   cout << a << " " << b << endl;
   swap(a, b);
   cout << a << " " << b << endl;

   cout << c << " " << d << endl;
   swap(c, d);
   cout << c << " " << d << endl;

   cout << e << " " << f << endl;
   swap(e, f);
   cout << e << " " << f << endl;

   cout << s1 << " " << s2 << endl;
   swap(s1, s2);
   cout << s1 << " " << s2 << endl;

   return 0;
}

// output:
//
// 2 3
// 3 2
// 4.5 6.7
// 6.7 4.5
// q r
// r q
// happy sad
// sad happy
```

```
// max.cpp
#include<iostream>
using namespace std;


// max(): maximum of two values of any type
template<typename T>
T const& max (T const& a, T const& b)
{
   return  a < b ? b : a;
}

// max(): maximum of three values of any type
template <typename T>
T const& max (T const& a, T const& b, T const& c)
{
   return ::max (::max(a,b), c);
}

int main()
{
   cout << ::max(7, 42, 68)     << endl;  // calls the template for three
   arguments
   cout << ::max(7.0, 42.0)     << endl;  // calls max<double> (by argument
   deduction)
   cout << ::max('a', 'b')      << endl;  // calls max<char> (by argument
   deduction)
   cout << ::max(7, 42)         << endl;  // calls max<int> (by argument
   deduction)
   cout << ::max<double>(7, 42) << endl;  // calls max<double> (no argument
   deduction)

   return 0;
}


// output:
//
// 68
// 42
// b
// 42
// 42
```

class Templates:

Can be "class"

Precede with:

```
template< typename T >
  :
```

e.g.

```
template< typename T >
class blah
{
   :
   // defns that use T
   :
}
```

Can declare different flavors of blahs

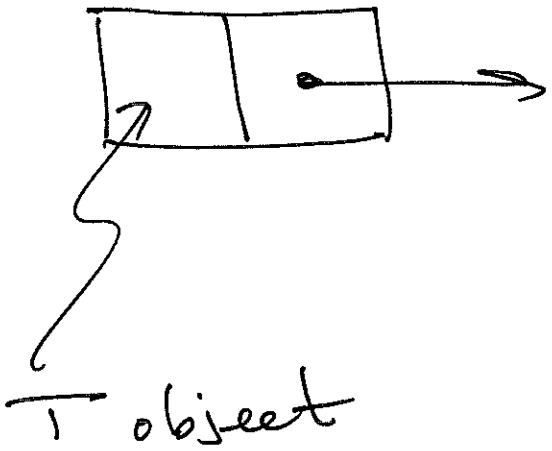
read: blah-of-int

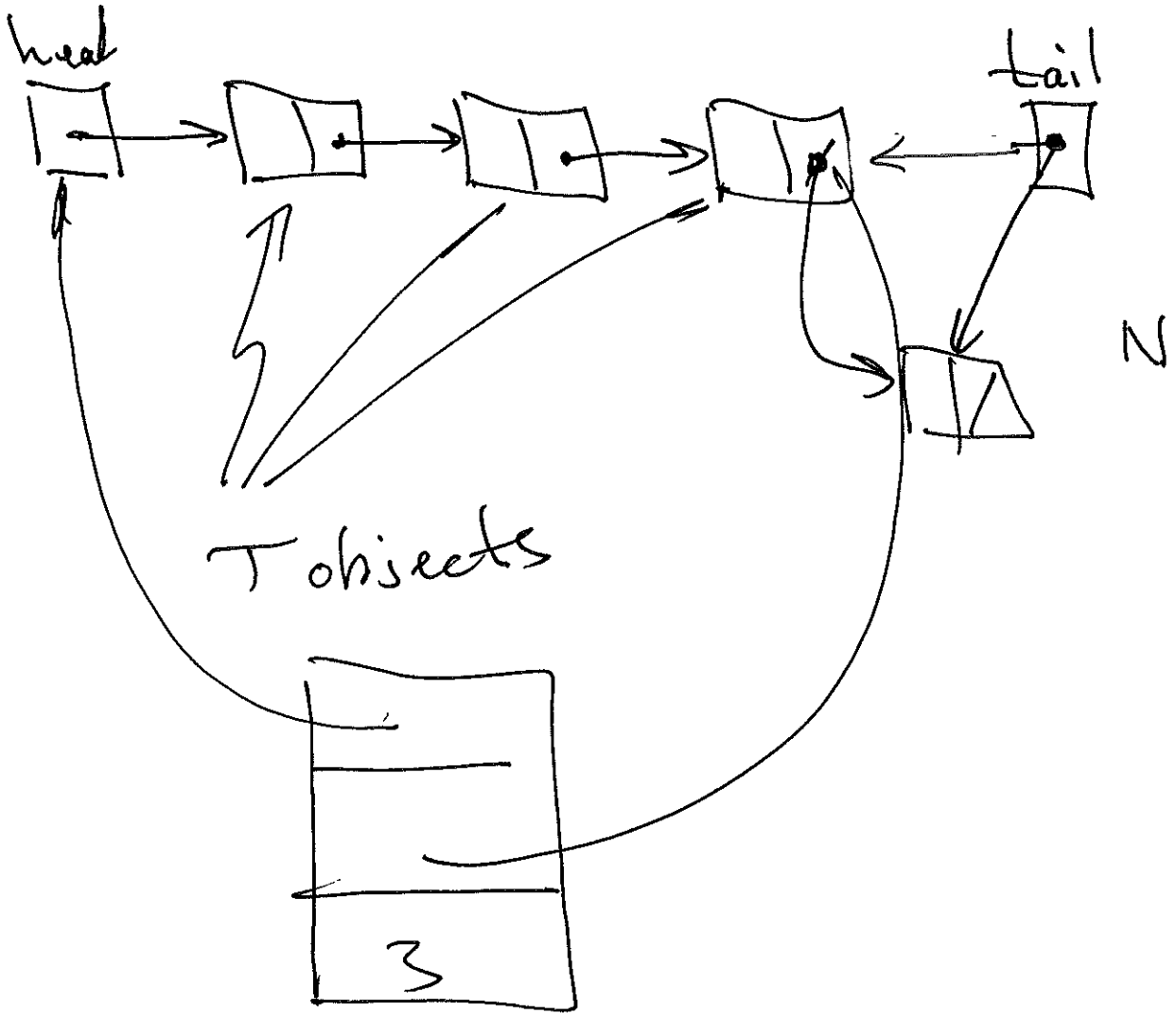
Ex.

```
blah<int> bi;

blah<double> bd;

blah<string> bs;

blah<blah<string>>  bbs;
```

-must- have space here

Node





```
// file: List.cpp
#include<iostream>
using namespace std;

// Node
template<typename T>
class Node
{
   public:
      Node();
      Node(T value) {data = value;};
      ~Node() {};

      // operations on Node<T>
      void setNext(Node<T>* N){next = N;}
      Node<T>* getNext(){return next;}
      T getData(){return data;}

   private:
      T data;
      Node<T>* next;
};

// List
template<typename T>
class List
{
   public:
      List() {length=0; head=0;};
      ~List() {};

      // operations on List<T>
      void appendToLast(Node<T>* pN);
      void deleteFirst();
      void print();

   private:
      int length;
      Node<T>* head;
      Node<T>* tail;
};


// List<T>::appendToLast()
// appends N to tail of list.
template<typename T>
void List<T>::appendToLast(Node<T>* N)
{
   if(length==0) // List is empty.
   {
      N->setNext(0);
      head=tail=N;
   }
   else
   {
      N->setNext(0);
      tail->setNext(N);
      tail=N;
   }
   length++;
}
```

tail = 0;

```
// List<T>::deleteFirst()
// deletes head of List
// Pre : length != 0
template<typename T>
void List<T>::deleteFirst()
{
   if(length==0) // List is empty
   {
      cout << "Error: calling deleteFirst() on empty List" << endl;
      exit(1);
   }

   if(length==1)
   {
      delete head;
      head = tail = 0;
   }
   else
   {
      Node<T>* N = head;
      head = head->getNext();
      delete N;
   }
   length--;
}

// List<T>::print()
// prints the data field for each Node
template<typename T>
void List<T>::print()
{
   Node<T>* N=head;
   int i;

   for(i=1; i<=length; i++)
   {
      cout << N->getData() << " " ;
      N = N->getNext();
   }
   cout << endl;
}


int main()
{
   List<string> L;

   L.appendToLast(new Node<string>("one"));
   L.appendToLast(new Node<string>("two"));
   L.appendToLast(new Node<string>("three"));
   L.appendToLast(new Node<string>("four"));
   L.deleteFirst();

   L.print();

   return 0;
}

// output:
//
// two three four
```