CMPS 109 2-28-11

- asg5 soon:
- asg4 linking errors: today
- website soon.

normally

module.h ← prototypes

module.cpp ← definitions
#include module.h

compile module.cpp to module.o

compilation unit

To use in main.cpp, #include module.h in main.cpp, compile to main.o, then Link using.

g++ prog main.o module.o

wont work if module.cpp contains templates (either fcn or class).

Why? template have not been instantiated at link time:

Ex. void fcn< typename T >(....){....}

What is T?

fcn<int>(...), fcn<string>(...)

That information is in main.cpp,

These two pieces of info:

- what to do (module.cpp)
- what kind of 'thing' to do it to (main.cpp)

lie in different files.

The Linker can't resolve this.

Common solution: inclusion model

Either

- Put everything in one file.

or

- Put all defns in module.h
  forget about module.cpp

or

- #include module.cpp at <u>end</u> of module.h (remove #include "module.h" from module.cpp.)

notice: all the same thing.

```
// main.cpp

#include "module.h"
```

```
////////////////////////////////////////////////////////////////////////////////
/
//
// file: List.h
// header file for the List ADT module
//
////////////////////////////////////////////////////////////////////////////////
/

#ifndef LIST_H_INCLUDE
#define LIST_H_INCLUDE

// Node<T> prototype
template<typename T> class Node;

// List<T> prototype
template<typename T> class List;

// Node<T> definition
template<typename T>
class Node
{
   friend class List<T>;

   public:
      Node(T x) {data = x; next=0;};
      ~Node() {};

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

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

      // operations on List<T>
      void appendToLast(T x);
      void deleteFirst();
      void print();

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

#include"List.cpp"

#endif
```

```
////////////////////////////////////////////////////////////////////////////
//
// file: List.cpp
// implementation file for the List ADT module
////////////////////////////////////////////////////////////////////////////
//

// List<T>::appendToLast()
// appends N to tail of list.
template<typename T>
void List<T>::appendToLast(T x)
{
   Node<T>* N = new Node<T>(x);

   if(length==0) // List is empty.
   {
      head=tail=N;
   }
   else
   {
      tail->next = N;
      tail=N;
   }
      length++;
}

// 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->next;
      delete N;
   }
   length--;
}

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

   while(N!=0)
   {
      cout << N->data << " " ;
      N = N->next;
   }
   cout << endl;
}
```

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

#include"List.h"

int main()
{
   List<string> L;

   L.appendToLast("one");
   L.appendToLast("two");
   L.print();
   L.appendToLast("three");
   L.appendToLast("four");
   L.deleteFirst();
   L.print();
   L.deleteFirst();
   L.deleteFirst();
   L.deleteFirst();
   // L.deleteFirst();  // error!
   L.print();
   L.appendToLast("five");
   L.print();

   return 0;
}



// output:
// one two
// two three four
//
// five
```

```
#############################################################################
#
#
#   Makefile for asg4
#
#############################################################################
#

prog: main.o
   g++ -o prog main.o

main.o : main.cpp List.cpp List.h
   g++ -c main.cpp

clean :
   rm -f prog main.o
```

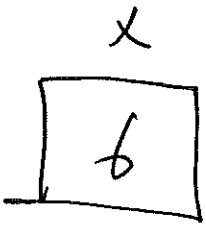
notes on 'const':

Simple ex

```
const int x;  // wrong
const int x = 6;  // right
```

Same as

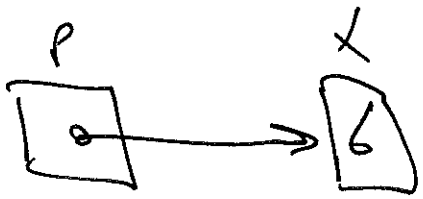
```
int const x = 6;
```



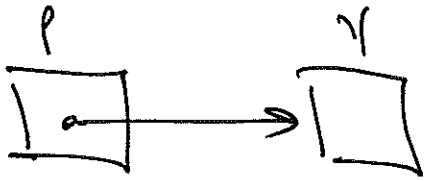
Ex `const int * p;`

Same as

```
int const * p;
p = &x;
```




```
int y = 7;
p = &y;
```



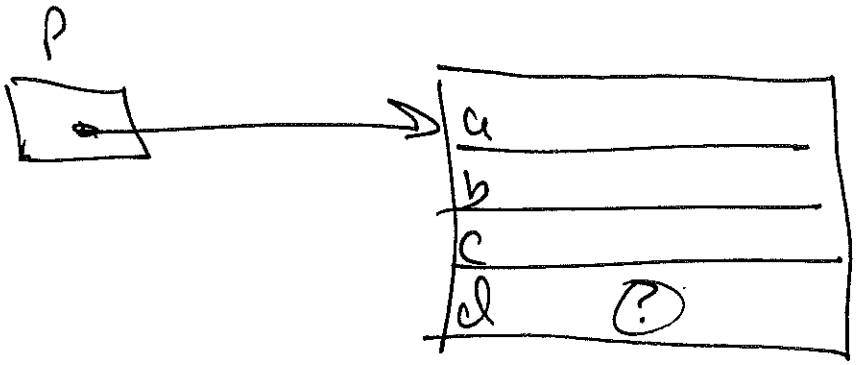

```
*p = 8; // error !!
```

note: { & address-of
* value-at

Given `int a = 3;`

```
int* q = &a
*&a == a; // true!
&*q == q; // true!
```

note -> OP:



(*p).d == p->d

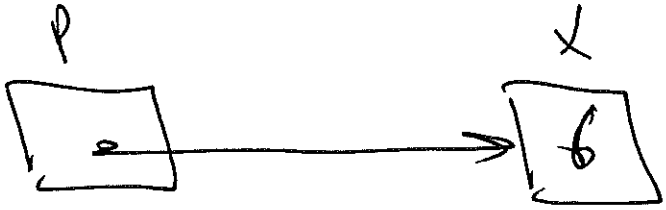
can define: given int x = 6;

int * p = &x; 00

int const * p = &x; 10

int * const p = &x; 01

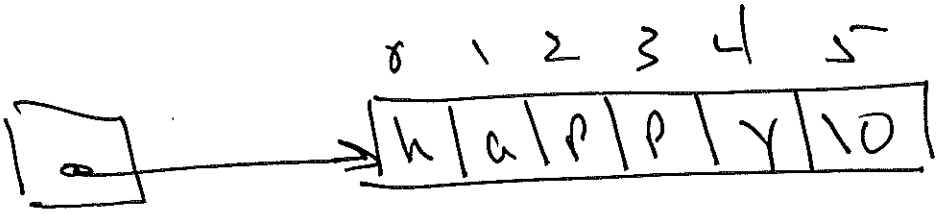
int const * const p = &x; 11



return values of fcns:

Ex.

```
char * fcn(){return "happy";}
```




```
fcn()[1] = 'o'; // runtime error
```

should have

```
char const * fcn(){return "happy";}
```

now `fcn()[1]='o';` // Compile-en
// error

Ex. `void g(big_struct_type const& p){....}`

reference

says & want change p

Ex.

```
class blah
{
    int var;
    void fcn( ) {.... }
}
```

Could be var = var+1;

Ex.

```
class blah
{
    int var;
    void fcn( ) const {.... }
}
```

bans fcn( ) from changing member variables.

Ex. in some class def'n

```
const int * const fcn(const int * const)
                                    const;
```

Same as:

```
int const * const fcn(int const * const)
                                    const;
```