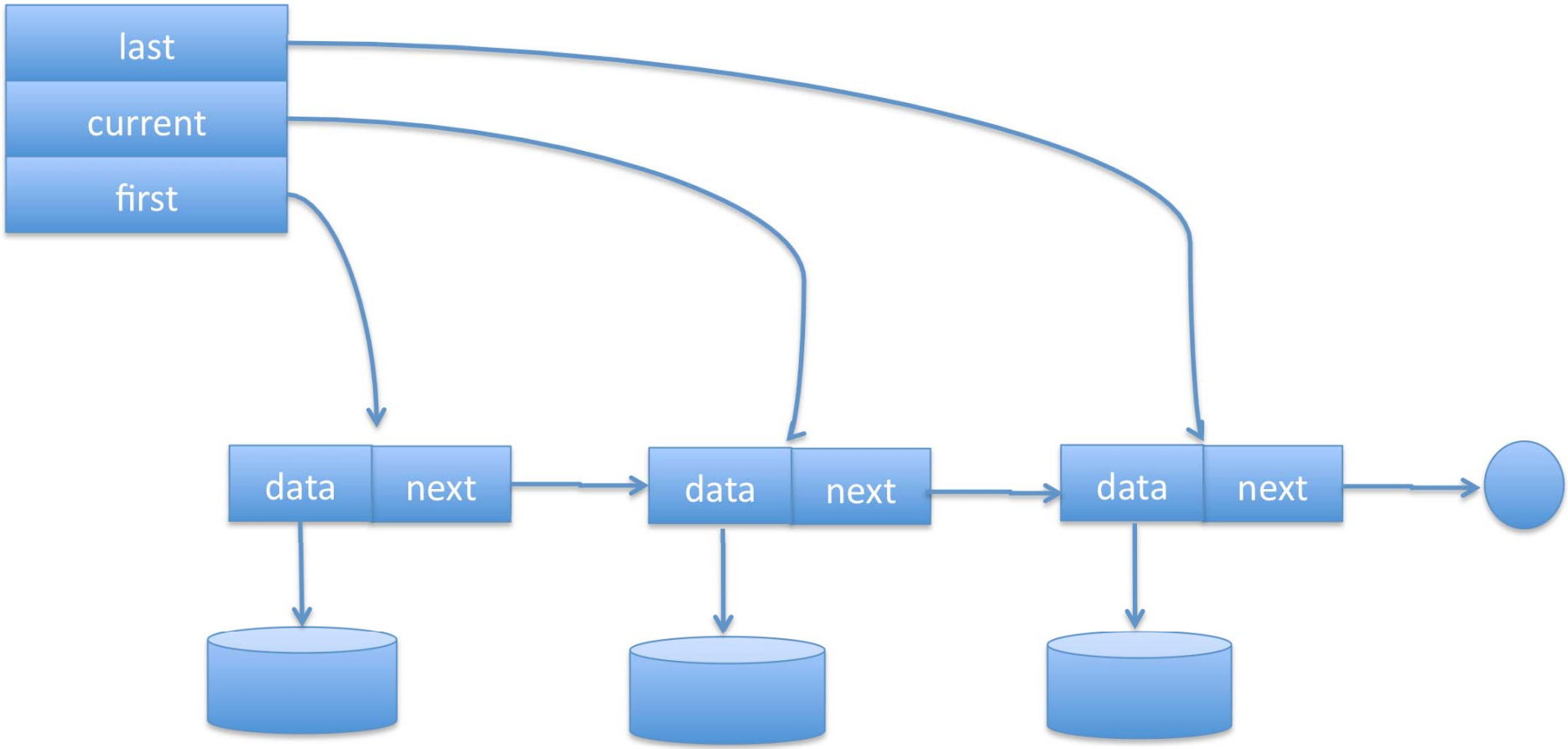
C++ Class Template

CpSc102 – Fall 2010

The List



delete void*

```
class list_t{
  private:
  struct link_t
  {
    link_t  *next;
    void    *data;
    link_t(void* idata=NULL) : next(NULL), data(idata){ };
  };
  link_t  *first;
  link_t  *last;
  link_t  *current;
  ...
}
```

```
void list_t::del_front_link(){
   link_t  *link;
   link = first;
   first = first->next;
   reset();
   delete link->data;
   delete link;
}
```

```
list.cpp: In member function 'void list_t::del_front_link()':
list.cpp:27: warning: deleting 'void*' is undefined
```

data_t

```
private:
  std::string   name;
  int  id;
```

data_t

```
//main.cpp
data_t* data=new data_t(name, num);
```

```
// constructors (overloaded)
data_t(std::string iname="",int iid=-1) : name(iname),  id(iid)
{ }

friend std::ostream& operator<<(std::ostream& s, const
    data_t& rhs)
{
    s << rhs.name << "  " << rhs.id << std::endl;
    return s;
}
friend std::ostream& operator<<(std::ostream& s, data_t
    *rhs)
{
     return(s << (*rhs));
}
```

```
//main.cpp
std::cout<<data;
```

list_t class template

```
template<typename T>
class list_t{
  private:
    struct link_t
    {
      link_t  *next;
      T   data;
       link_t(T idata=NULL) : next(NULL), data(idata) { };
    };
    link_t  *first;      // first link in the list
    link_t  *last;       // last link in the list
    link_t  *current;

  public:
    ...
}
```

Constructors, Assignment Operator

```
list_t() { first = last = current = NULL; }

list_t(const list_t& rhs) : \
first(rhs.first),  last(rhs.last), current(rhs.current)
{ }

const list_t& operator=(const list_t& rhs)
{
    if(this != &rhs) {
        first = rhs.first;
        last = rhs.last;
        current = rhs.current;
    }
}
```

```
//main.cpp
list_t<data_t *>    list;
```

list_t

```
void  reset()      { current = first; }

int   empty()     { return first == NULL ? 1 : 0; }

int   not_end()   { return current != NULL ? 1 : 0; }

int   next_link()     { current = current->next; }

T get_data()      { return(current->data); }
```

add an element to end of a list

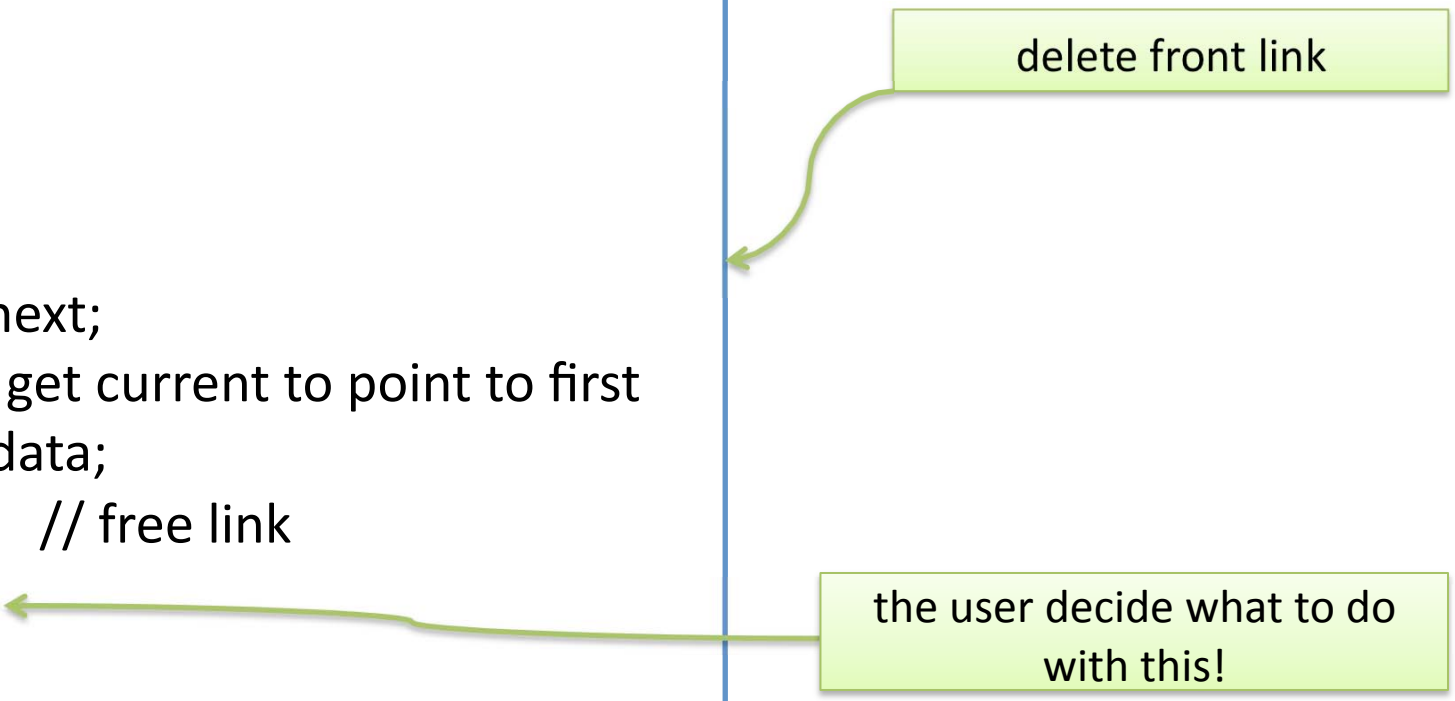
```
//list.cpp
template <typename T>
void list_t<T>::add(T data)
{
      link_t  *link = new link_t(data);

      // add an element to end of a list
      if(empty()) {
        first = link;
        last = link;
      } else {
        last->next = link;
        last = link;
      }
}
```

Lab7

```
void list_t::add(void *data)
{
    link_t  *link = new link_t(data);
    ...
}
```

```
template <typename T>
T list_t<T>::del_front_link()
{
    link_t  *link;
    T   data;
    link = first;
    first = first->next;
    reset();      // get current to point to first
    data = link->data;
    delete link;      // free link
    return data;
}
```



delete all list structures and items

```
template <typename T>
void list_t<T>::del()
{
    T   data;
    reset();

   // delete all list structures and items
   while(not_end()) {
         data = del_front_link();
         delete data;
     }
}
```

Specializations

```
//list.cpp
template class list_t<data_t *>;
```

```
//main.cpp
list_t<data_t *>    list;
data_t         *data=NULL;
std::string     name;
int         num;
while(!std::cin.eof()) {
        std::cin >> name >> num;
        if(std::cin.good()) {
           data = new data_t(name,num);
           list.add(data);
        }
}

list.reset();
while(list.not_end()) {
        data = list.get_data();
        std::cout << data;
        list.next_link();
}

list.reset();
while(list.not_end()) {
        data = list.del_front_link();
        delete data;
}
```


```
//main.cpp
list_t list;
data_t *data=NULL;
std::string name;
int num;
while(!std::cin.eof()) {
        std::cin >> name >> num;
        if(std::cin.good()) {
        data = new data_t(name,num);
        list.add((void *)data);
        }
}
...
```

```
//main.cpp
list_t<data_t *>   list;
data_t        *data=NULL;
std::string    name;
int        num;
while(!std::cin.eof()) {
       std::cin >> name >> num;
       if(std::cin.good()) {
          data = new data_t(name,num);
          list.add(data);
       }
 }

 list.reset();
 while(list.not_end()) {
       data = list.get_data();
       std::cout << data;
       list.next_link();
 }

list.reset();
while(list.not_end()) {
       data = list.del_front_link();
       delete data;
 }
```

Question: what if we use list_t<data_t> list ?

Makefile

```
CC  = g++
INCLUDE = -I.
CFLAGS  = -g -m32
LDFLAGS = -L.  -L/usr/lib

LDLIBS  = -list  -lc -lm

.cpp.o:
   $(CC) -c $(INCLUDE) $(CFLAGS) $<

OBJS = data.o

all: libist main

libist: libist.a
libist.a: list.o
   ar rcs $@ $?
   ranlib $@

main: main.cpp main.o $(OBJS) libist.a
   $(CC) $(CFLAGS) $(INCLUDE) -o $@ $@.o $(OBJS) $(LDFLAGS) $(LDLIBS)
```