Data Abstraction

Process of
- Creating our own data types
- User Defined Data Types (UDT)
- Abstract Data Types (ADT)
  (is a Data Type we've created)

- List ADT

Defining the List Class

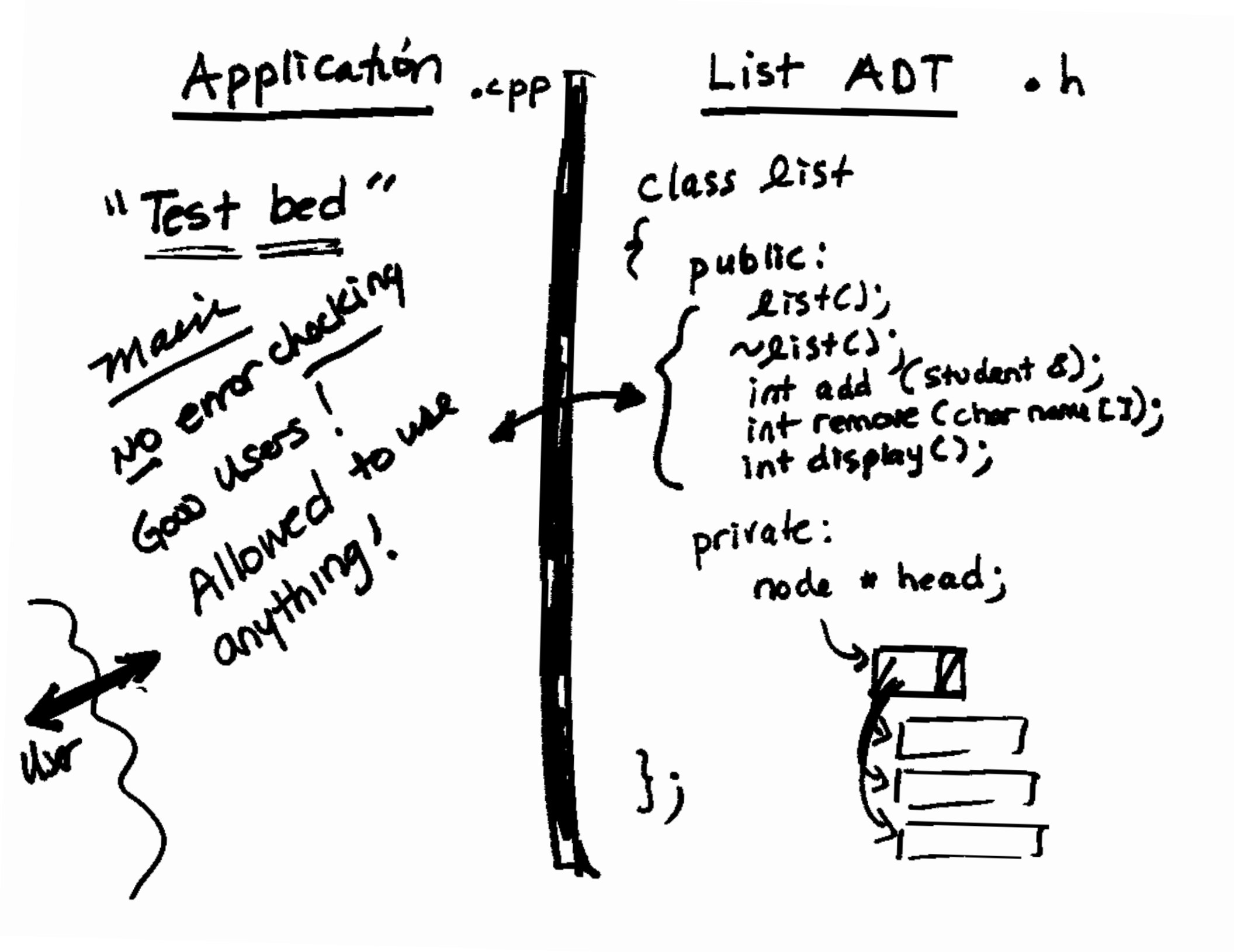
```
class list {
  public:
     list();
     ~list();
     int insert(const data &);
     int retrieve(char *, data &);
     int display();
     int remove (char *);
  private:
     node * head;
};
```

student

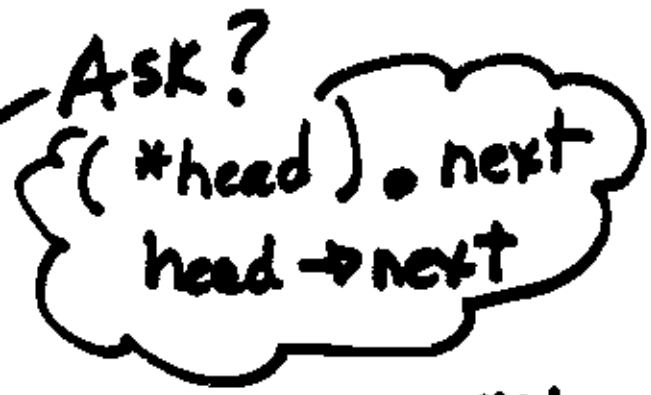
Student

```
struct student
{
    char *name;
    char *psu;
    char *addr;
    char *email;
      ⋮

};
struct node
{   student [redacted] a_student;
    node * next;

};
```

All Arrays dynamically allocated


Application .cpp
List ADT .h
"Test bed"
Main
No error checking
Good users!
Allowed to use anything!
User
class list
{
public:
list();
~list();
int add (student s);
int remove (char name[]);
int display();
private:
node * head;
};

```
list :: list()
{   head = NULL;
}
list :: ~list()
{   node * temp;
    while (head)
    {   temp = head -> next
        delete [] head -> a.student.name;
        delete [] head -> a.student.pan_id;
        delete head;
        head = temp;
    }
}
```



```
int list::insert(const student &s)
{
  if (!head)
  {
    head = new node;
    head->next = NULL;
    head->a_student.name = new char[strlen(s.name) + 1];
    strcpy(head->a_student.name, s.name);
  }
}
```

Insert or Add

```
node * current = head;
while (current && strcmp(S.name, current->a_student.name) > 0)
{
    previous = current;
    current = current->next;
}
```

```
class student
{
    public:
        student();
       ~student();
        int set_all (student &);
        int reset ()
        student (const student &);
        student(char * name);
    private:
        char * name;
        char * psuid;
        char * email;

};
```

```
int list::add (student & s)
{
    :
  // Traverse ....
  head = new node;
  head->next = NULL;
  head->a_student.setall (s);
```

head->a_student.setall(s);

```
list :: ~list()
{
    node * temp;
    while (head)
    {   temp = head -> next;
        delete head;
        head = temp;
    }
}
```

```
student :: ~student()
{
    delete [] name;
    delete [] psuId;
    delete [] email;
}
```

list class

Insert

```
node * current = head;
while ( current && current->a_student.compare(s))
{   previous = current;
    current = current->next;
}
```

True when s's name smaller !

```
int student :: compare (student & s)
{
  if( strcmp (name, s.name) < 0) return 1;
  return 0;
}
```