Hayden Fuller 662028619 Section 5 TA: Charles Mentors: Hanson & Theodore & Ali,LecTF 12-1150,LabWed2-350,hwF11a,autoM10a,ppM1p,TutoringMT6,W4,T12,F4-6,ALAC,dropIn 8-10M-T 5,0-10,8-15-29,20h-33,20n-16,tm-5

```
sizeof() gives memory, the means the space a
pointer takes up if given
Int = float will truncate, does not work in
vectors
#include… iostream, cmath, cstdlib (exit),
string, vector, fstream, algorithm, map
cctype for isdigit, is alpha, tolower
bool isThisTrue(int same, double &change,
std::string &large) { return true; }
#include <utility> std::pair<int,int> f;
f=std::make_pair(10,20);
int main(int argc, char* argv[]) {
  std::cout << “enter something” <<
std::endl;
  std::cin >> something1 >> something2;
  std::cerr << “wrong something”
  int a[8];  a[2] = 16;
  std::string stars(12,’*’);
  stars2 = stars.c_str();
  char h[] = “HW!”; //OR {‘H’, ‘W’, ‘!’,
‘\0’}; std::string h2(h); //copy cstr
  std::vector<int> v(10, 5);
  std::vector<itn> c(v); //copy
std::sort(c.begin(),c.end(),optional);//no()
  for(int i = 0; i < 2; i++) { while(1) {
break; } }
  std::ifstream file_in(“input.txt”);
  std::ofstream file_out(“output.txt”)
  if(!file_in.good()) { std::cerr << error;
exit(1); }
  file_in >> s >> s1;    file_out << h2 <<
“hello”;
  Int x; while(file_in >> x) {v.push_back(x)}
  const int n = 10; int *p = &x;  int a[n];
  for(p=a; p<a+10; p++){ *p=sqrt(p-a);}
  int *a = new int[n]; for(int *p = a; p<a+n;
p++){
  int** a = new int*[r];  for(int i=0;
  i<r;i++){a[i]=new int[c]; for(int
  j=0;j<c;j++){a[i][j]=int(i+1)/int (j+1);}}
  int readInt; int* intArray=new int[max];
  while(input>>readInt){*(intArray +
*numElements) = readInt; *numElements += 1;}
  str.substr(start, length);   //npos = end
of string
  str.find(“world”);  //returns iterator of
place
  //students.push_back(Student(name, age));
  Student stu(“name”, 19);
  std::cout << stu << std::end;
  return 0; }
```

```
#ifndef __student_h_#define __student_h_
#include …
class student {
public:
  Student();
  Student(std::string aName, int aAge);
  std::string getName() const;
  int getAge() const;
  void setName(std::string aName);
  void setAge(int aAge);
  bool sameName(const Student& s2) const;
  void print() const;
private:
  std::string name; int age;
}
bool operator< (const Student& s1, const
Student& s2);
std::ostream& operator<< (std::ostream&
ostr, const Student& s);
#endif

In student.cpp
#include…   #include “student.h”
Student::Student() {  name = “No-name”; age
= 0; }
Student::Student(std::string aName, int
aAge) { name = aName; age = aAge;}
Std::string Student::getName() const
{ return name; }
int Student::getAge() const { return age; }
void Student::setName(std::string aName)
{ name = aName; }
void Student::setAge(int aAge) {age=aAge;}
bool Student::sameName(const Student& s2)
const { //check }
bool operator< (const Student& s1, const
Student& s2) { /* sort */ return true; }
std::ostream& operator<< (std::ostream&
ostr, const Student& s) { ostr <<
s.getName() << “ - “ s.getAge() <<std::endl;
return ostr; }
a->b is shorthand (*a).b use it when getting
something from the pointer thing
template <class T> class Vec {
public: typedef T* iterator;
typedef const T* const_iterator;
typedef unsigned int size_type;
Vec() { this->create(); }
Vec(size_type n, const T& t = T()) {
this->create(n, t); }
Vec(const Vec& v) { copy(v); }
Vec& operator=(const Vec& v);
```

```
~Vec() { delete [] m_data; }
T& operator[] (size_type i) { return
m_data[i]; }
const T& operator[] (size_type i) const {
return m_data[i]; }       void
push_back(const T& t); iterator
erase(iterator p);
void resize(size_type n, const T&
fill_in_value = T());
void clear() { delete [] m_data;  create();}
bool empty() const { return m_size == 0;}
size_type size() const { return m_size;}
iterator begin() {return m_data;}
const_iterator begin() const {return
m_data;}
iterator end() {return m_data+m_size;}
const_iterator end() const {return
m_data+m_size;}
private:
void create();
void create(size_type n, const T& val);
void copy(const Vec<T>& v);
T* m_data; size_type m_size;
size_type m_alloc;};
template <class T>class Node {public:
Node():next_(NULL),prev_(NULL){}
Node(const T& v):value_(v),next_(NULL),
prev_(NULL){} T value_;Node<T>* next_;
Node<T>* prev_;};
template <class T>class list_iterator
{public:list_iterator():ptr_(NULL){}
list_iterator(Node<T>* p):ptr_(p){}
list_iterator(const list_iterator<T>&
old):ptr_(old.ptr_){}
list_iterator<T>& operator=(const
list_iterator<T>& old){ptr_=old.ptr_;
return *this;}  ~list_iterator(){}
T& operator*()  { return ptr_->value_;  }
list_iterator<T>& operator++() {
ptr_=ptr_->next_;return *this;}
list_iterator<T> operator++(int) {
list_iterator<T> temp(*this);
ptr_ = ptr_->next_;return temp;}
list_iterator<T>& operator--() {
ptr_ = ptr_->prev_;return *this;}
list_iterator<T> operator--(int) {
list_iterator<T> temp(*this);
ptr_ = ptr_->prev_;return temp;}
friend class dslist<T>;
bool operator==(const list_iterator<T>& r)
const { return ptr_ == r.ptr_; }
bool operator!=(const list_iterator<T>& r)
const { return ptr_ != r.ptr_; }
private: Node<T>* ptr_; };
template <class T>
bool binsearch(const std::vector<T> &v, int
low, int high, const T &x) {
if (high == low) return x == v[low];
int mid = (low+high) / 2;
if(x<=v[mid]){return binsearch(v,low,mid,x);
}else{return binsearch(v, mid+1, high, x);}
template<class T>void mergesort(vector<T>&va
lues){vector<T>scratch(values.size());merges
ort(0,int(values.size()-1),values,scratch);}
template <class T>void mergesort(int low,int
high,vector<T>& values,vector<T>& scratch) {
if(low>=high)return; int mid=(low+high)/2;
mergesort(low, mid, values, scratch);
mergesort(mid+1, high, values, scratch);
merge(low, mid, high, values, scratch);}
template<classT>void merge(int low,int mid,
inthigh,vector<T>&values,vector<T>&scratch){
int i=low, j=mid+1, k=low;
while(i<=mid&&j<=high){
if(values[i]<values[j]){
scratch[k]=values[i];++i;
}else{scratch[k]=values[j];++j;}++k;}
for(;i<=mid;++i,++k){scratch[k]=values[i];}
for(;j<=high;++j,++k){scratch[k]=values[j];}
for(int l=low;l<=high;++l)
values[l]=scratch[l];
itr begin() end() insert(itr, v) erase(itr)
int size() bool empty() front() back()
push_back(v) pop_back() clear()
Vec: [i] at(i)
List: push_front(v) pop_front() sort(isLess)
map<string,int> count; ++count[s];
map<string,int>::const_iterator it;
it->first, second
pair<string,int> p=make_pair("s", 1
set<string> list; pair=list.insert("h");
it location=pair.first;bool added=p.second;
Tree:inOrder:left-right,ignore vertical.
preOrder:startRoot,downLeft,upDown
postOrder:startLeft,leavesUp
class hash_string_obj{public:unsigned int
operator()(const std::string& key)const{un
int hash=1315423911;
for(unsigned int i = 0; i < key.length();
i++)hash^=((hash<<5)+key[i]+(hash>>2));
return hash;}};
ds_hashset<string,hash_string_obj> hs;
unordered_map<string,int,optdecltype(&functi
on)ORhash_string_obj> m(opt? size)
for_each(vec.begin();vec.end(),int_print);
```