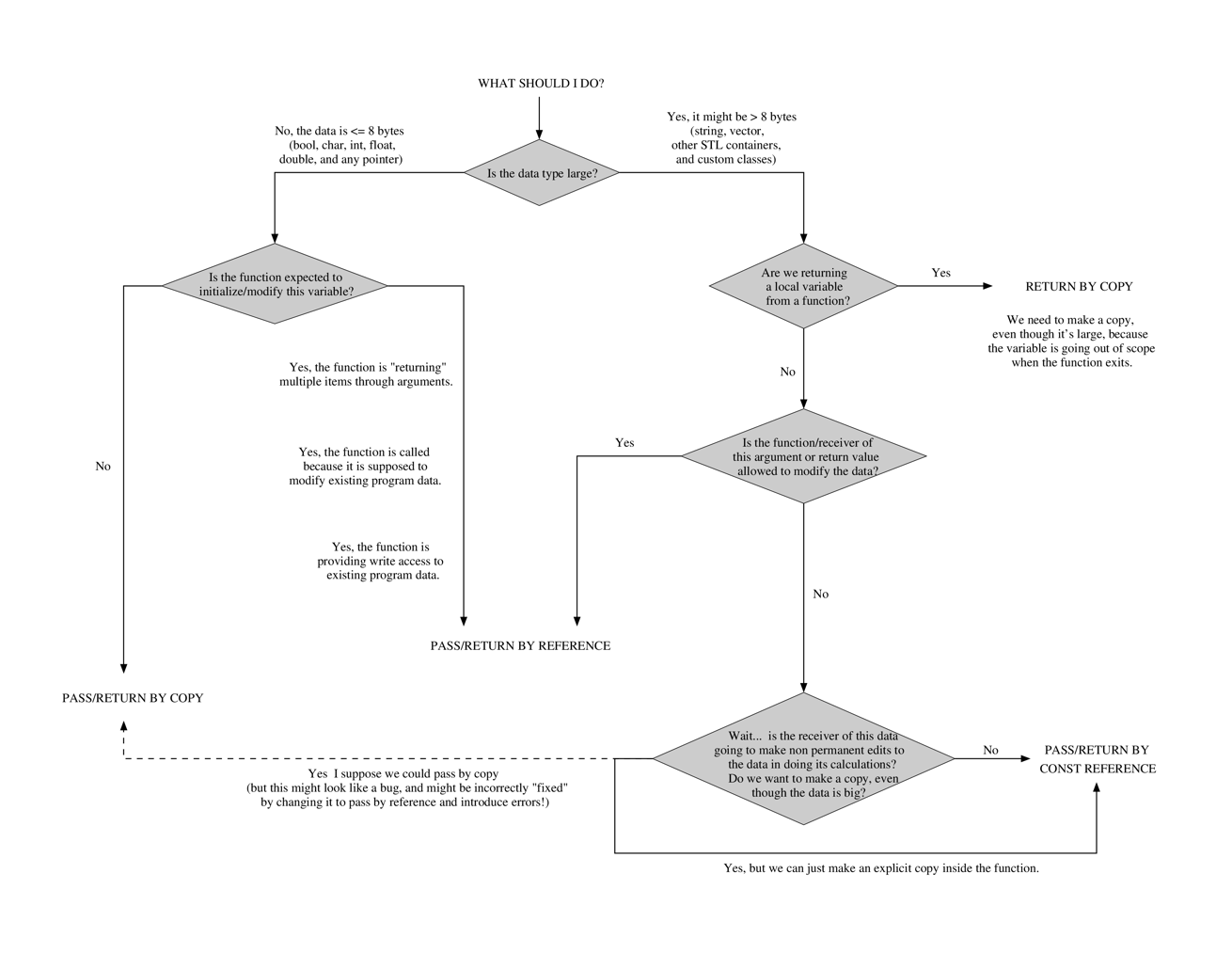
Jared Gridley Test 2

TA: Guanghan/Francis Mentors: Fred, Rutvik, Zach

**A screenshot of a cell phone

Description automatically generated**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **STL Vector/ Vec<T>** | **STL List/ dslist<T>** | **Singly-LL** | **Doubly-LL** |
| **O(n)** | Size: O(1)  push\_back: O(1)  erase: O(n)  insert: O(n)  pop\_back: O(1)  resize: O(n)  operator=: O(n)  clear: O(n)  sort: O(nlog(n)) | Size: O(1)  push\_back/push\_front: O(1)  erase: O(1)  insert: O(1)  pop\_back/pop\_front: O(1)  resize: O(n)  operator=: O(n)  clear: O(n)  sort: O(nlog(n)) | Size: O(n)  push\_back: O(n)  erase: O(1)  insert: O(1)  pop\_back: O(n) | Size:  push\_back:  erase:  insert:  pop\_back: |
| **Initialize** | std::vector<int> vname; | std::list<int> Lname | Node\_class \*head = NULL; | |
| **Insert** | vname.insert(itr\_i, val) | Lname.insert(itr\_i, val) | Insert B into T:  B->next= T->next;  T->next = B; | B->prev = T;  T=t->next;  B->prev->next = B;  T->prev = B;  T->prev->next = T; |
| When iterating: returns iterator to same place in list, but really the next element:   * name.insert(itr—) * name.erase(itr--) | |
| **erase** | Vname.erase(itr\_position) | Lname.erase(itr\_position) | P = T;  T = T->next  P->next= T->next  delete temp | P = T  T = T->next;  P->next = T->next  P->next->prev=P  delete T |
| Erase can invalidate the iterator. | Use instead of vector (if a lot) |
| **Push\_back** | Sometimes alloc new array size 2\*m\_alloc | .Push\_back(value)  🡨 | P->next = T;  T->next = NULL | P->next = T;  P->next->prev = P  T->next = NULL; |
| **Pop\_back** | Vname.pop\_back() | Lname.pop\_back() | P = T  T = T->next  P->next = NULL | P = T  T = T->next  P->next->prev = NULL  P->next = NULL |
| Pop\_back/push\_back/Insert/Erase -> First iterate to the point you want to remove **-1** | |
| Does not return value | |
| **Iterators** | Vector<int>::iterator v\_itr = v.begin()  Has []  Uses operator<  vector<T>::insert() returns an itr b/c insert() may invalidate le itr passed in | list<int>::iterator l\_itr = l.begin()  No []  Uses operator<  Incrementing the end() iterator in any STL list has undefined behavior | N/A | |

**Recursion:**

* Infinite recursion can == segmentation fault

|  |  |
| --- | --- |
| void fun(int n){  if(n>0){  print(n)  fun(n-1);  }  } | 5 4 3 2 1 |
| void fun(int n){  if(n>0){  fun(n-1);  print(n);  }  } | 1 2 3 4 5 |

|  |  |
| --- | --- |
| Sorting Time Complexities | |
| Binary | O(log(n)) |
| Merge | O(nlog(n)) |
| Insertion | O(n^2) |
| Bubble | O(n^2) |
| Recursive is the same | |

Classes:

Declaration 🡪 Class object\_name( argument );

Class Date{

public:

//Constructor

Date();

Date(int aMonth, int aDay, int aYear);

//Accessor

int getDay() const;

//Modifier

void setday(int bday);

//Other member functions

bool isEqual(const Date& \*date2) const;

int lastDayInMonth() const;

void print() const;

Private:

int day;

};

Implementation:

bool Date::isLeapYear() const{

}

Pointers:

* Using operator[] on a pointer is the same as using pointer arithmetic and then dereferencing the result.
* Writing int\* x=5; will result a compiler error.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Maps | Sets | Ropes | Binary Search Tree | Hash Tables (LL) |
| **O(n)** |  |  |  |  |  |
| **Initialize** |  |  |  |  | \*have a pre-defined class |
| **A close up of text on a black background  Description automatically generatedInsert** |  |  |  |  | 1.Check for Resize  2. Convert hash value and then table index.  3. Find key (if in table)  4. if not, add it to table  5. else, do not insert (or LL) |
| **Erase** |  |  |  |  | 1. Iterator to find key.  2. if found erase(p)  3. else do nothing (return) |
| **Iterators** |  |  |  |  | RESIZE the table |
| **operator++** |  |  |  |  | Move to the next element in the “table”  iterator& op++()  this->next();  return \*this |
| **operator--** |  |  |  |  | Move to prev element in table.  iterator& op—()  this->prev()  return \*this |
|  |  |  |  |  |  |