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

Crib Sheet: Test 1

## →Declare the pointer before creating heap alloc:

```
Float* oat[3];
Oat[1] = new float; *oat[1] = 3.14;
Oat[2] = new float; *oat[2] = 6.02;
Float rice;
Float *wheat;
Wheat = oat[2];
Float **barley = new float*;
*barley = oat[1];
```

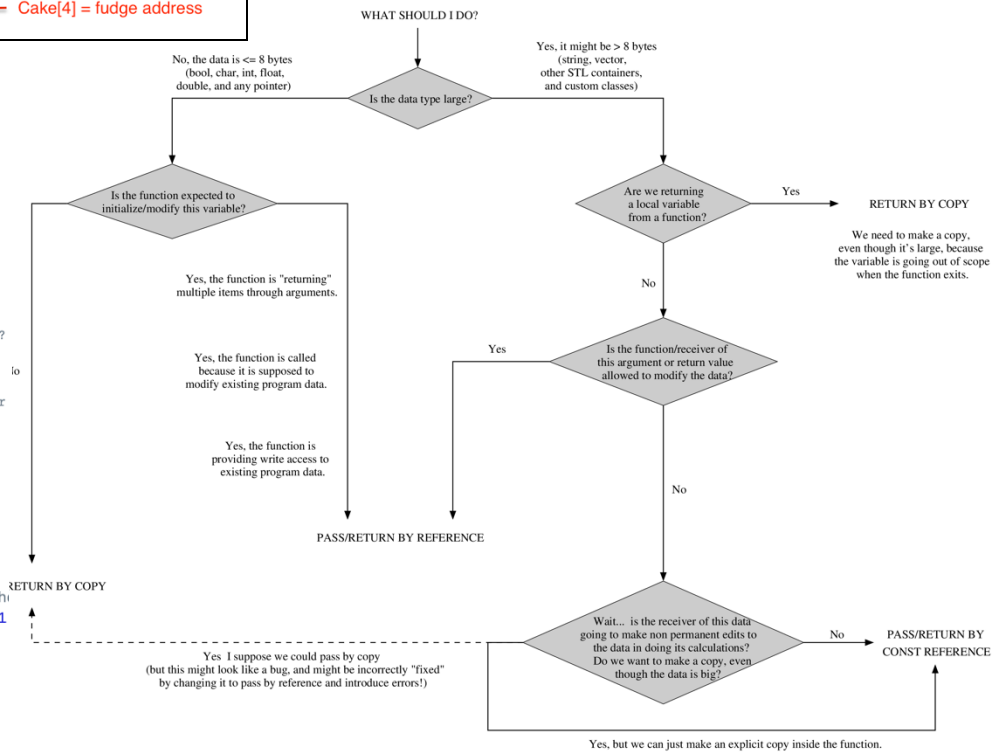
```
Bool** cake;          Pointer to pointer
Bool pie; Bool fudge;
Pie = true;           Setting stack values
Cake = new bool*[5];  Cake = Arr of ptrs in heap
Cake[1] = &pie;       Cake[1] points to pie address
Bool* donut = new bool;Donut in stack points to heap
*donut = false;       Change donut in heap
Cake[2] = donut;      Cake[2] = donut value
Cake[4] = &fudge;     Cake[4] = fudge address
```

DYNAMIC MEMORY: DELETES
If you have an array in a heap array, delete the array in the array before the heap array.
*In 2-D arrays → Delete all the arrays and then the outer array.

## Classes:

```
class Date {
public:
  //Constructors
    Date();
    Date(int aMonth, int aDay, int aYear);
  // ACCESSORS
    int getDay() const;
    int getMonth() const;
    int getYear() const;
  // MODIFIERS
    void setDay(int aDay);
    void setMonth(int aMonth);
    void setYear(int aYear);
    void increment();
  // other member functions that operate on date objects
    bool isEqual(const Date& date2) const;  // same day, month, & year?
    bool isLeapYear() const;
    int lastDayInMonth() const;
    bool isLastDayInMonth() const;
    void print() const;                     // output as month/day/year
private:  // REPRESENTATION (member variables)
    int day;
    int month;
    int year;

};
```




## Implementation:

```
#include <iostream>
#include "date.h"
// array to figure out the number of days, it's used by th
const int DaysInMonth[13] = {0, 31, 28, 31, 30, 31, 30, 31
Date::Date() { //default constructor
    day = 1;
    month = 1;
    year = 1900; }
int Date::getDay() const {
    return day;
}
bool Date::isLeapYear() const {
  return (year%4 ==0 && year % 100 != 0) || year%400 == 0;
}
```

## Dynamic Memory (2-D)

```
double** a = new double*[rows];
for (int i = 0; i < rows; i++) {
  a[i] = new double[cols];
  for (int j = 0; j < cols; j++) {
    a[i][j] = double(i+1) / double (j+1);
  }
}
```

**Vectors:**
Declare: std::vector < double > vector_name;
- Vectors can only be copied to ones with the same type.

Vector_name.size(); //Capacity
Vector_name.begin(), vector_name.end(); //Iterator
Vector_name.push_back(value) //Modifiers
Vector_name[ ]; //Accessor

## Memory Debugging

- Use of uninitialized memory
- Reading/writing memory after it has been free'd (*NOTE: delete calls free*)
- Reading/writing off the end of malloc'd blocks (*NOTE: new calls malloc*)
- Reading/writing inappropriate areas on the stack
- Memory leaks - where pointers to malloc'd blocks are lost forever
- Mismatched use of malloc/new/new [] vs free/delete/delete []
- Overlapping src and dst pointers in memcpy() and related functions

**Creating a new object:**
Class object_name( argument )

**WHEN TO USE "const":**
Declaring a function in .h file (and subsequentially in implementation file) → After function ( ).

Member functions that do not change member variables (passing by argument) → before the class name

Ex:
bool Date::isEqual (const Date &date2) const;

## Memory Diagrams

Consider the following code:

```
int i,**j,k,l,*m;
i = 0;
j = new int*[3];
j[0] = new int;
j[1] = &i;
m = *(j+1);
j[1] = &k;
k=10;
*(j[0]) = 5;
j[2] = j[0];
*(j[0]) = 18;
*m = 4;
l = 3;
```