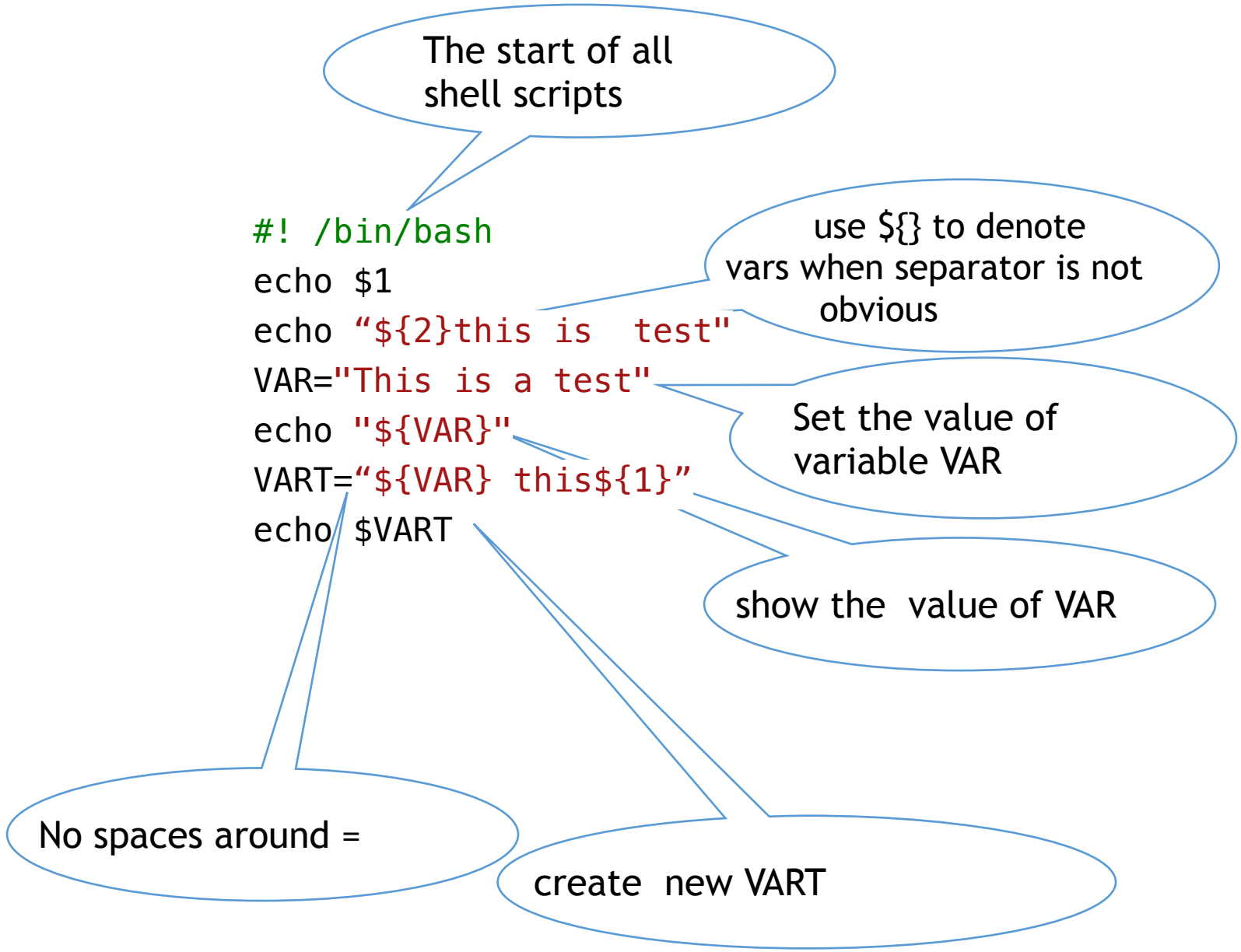
CS246
Unix: submit
C:more dynamic memory
linked lists and queues

April 6

Shell scripts

- language based on algol
  - an otherwise dead PL
- $1, $2, … command line args
- VERY sensitive to whitespace



Shell script if statements

- space before and after [and]
- then on separate line

- Comparisons
  - numbers
    - -gt -lt -eq
  - Strings
    - -z -n (empty and not empty)
    - = != (equal and not equal

- [ ] syntax actually just invokes the unix “test” command so can directly check effect of any [ ]
  - UNIX> test 50 -gt 100
  - UNIX> echo $?
    - prints 1
      - that is false recall that UNIX 0 is good!

```
#!/bin/bash
# Comment about Basic if statement
VAR="Hello"
if [ $1 -gt 100 ]
then
echo Hey that\'s a large number.
VAR=`pwd`
fi
echo "VAR $VAR"
```

more shell script if

- Files in if
  - -d exists and is directory
  - -e exists
  - -r exists and readable
  - -s exists and non-zero size
  - -w exists and writable
  - -x exists and executable

```
#!/bin/bash
if [ -s $1 ]
then
  echo "$1 is here and not empty"
else
  if [ -e $1 ]
  then
    echo "$1 is here but empty"
  else
    echo "$1 is not here"
  fi
fi
```

The submit script

- to set variables
  - XXX=…
    - e.g. DATE
    - var names Convention UPPER CASE
    - NO SPACE around =
- To use variables
  - $XXX
  - ${XXX} also works
- if [$? …
  - $? holds the result of the last unix command
    - not output, result
  - spaces are important
  -

```
echo "Submitting Project ${PROJECT} for CMSC${COURSE} with ${PI
PROJDIR=/home/${PROF}/submissions/${SEMESTER}/cmsc${COURS
if [ ! -d $TARGET ]; then
        echo "Target is not a directory."
        echo "Specify a directory containing your assignment afte
        exit 1
fi
DATE=`date +%F-%H-%M-%S-%Z`
TARNAME=${USER}-project${PROJECT}-${DATE}.tar
GZNAME=$TARNAME.gz
echo "Creating archive for submission...
tar cvfz $GZNAME $TARGET
echo "\nSubmitting archive..."
cp $GZNAME $PROJDIR/
rm $GZNAME $TARNAME
if [ ! $? -eq 0 ]; then
        echo "Submission failed! Please correct any errors and try
        exit 1
else
        echo "Submission complete! Submission timestamp is $DAT
fi
```

existence of a directory

`` == execute unix command

the result of the most recent unix command

recall that 0 is good

putting malloc, free and structs together

- Reading the text file into minimal space
  - does require 2 reads of the the file
- could pipe wc but that would still read the entire file.

- Note. Since the array and its contents were all malloc'd, they must all be free'd.
  - be sure to free contents before freeing array.

```
int main(int argc, char* argv[]) {

    FILE* f = fopen(argv[1], "r");
    if (!f) {
        fprintf(stderr, "No such file\n");
        return 1;
    }
    fclose(f);

    int linecount = linecounter(argv[1]);
    char** text = readfile(argv[1], linecount);
    for (int i=0; i<linecount; i++)
        printf(text[i]);

    for (int i=0; i<linecount; i++)
        free(text[i]);
    free(text);

    fclose(stdin);
    fclose(stdout);
    fclose(stderr);
}
```

Applying all of this to Weather

- Core idea
  - for every struct have a constructor and destructor
  - constructor allocates space
  - destructor frees
- **Always** use constructor to get struct
  - That way the destructor can always work.

Weather wind

file wwind.h

```
typedef struct {
    char * direction;
    int speed;
    char * scale;
} Wind;

Wind* makeWind(char* dir, int sp,
char* scl);
void freeWind(Wind* wnd);
```

Constructor

```
#include "wutil.h"
#include "wwind.h"
#include <stdlib.h>

Wind* makeWind(char* dir, int sp, char* scl) {
    Wind *rtn = malloc(1 * sizeof(Wind));
    rtn->direction = strmcopy(dir);
    rtn->speed = sp;
    rtn->scale = strmcopy(scl);
    return rtn;
}

void freeWind(Wind* wnd) {
    free(wnd->direction);
    free(wnd->scale);
    free(wnd);
}
```

Destructor

utility functions

- Used by multiple .c files.
- I usually put these into files named util.[ch]
- Not affiliated with a single struct

file: wutil.c

```
#include <string.h>
#include <stdlib.h>

/**
 * Create a copy of the provided string in a newly malloc'd
 * block of memory.  The block is exactly the size needed for
 * the copy.  THIS MUST BE FREED
 * @param scr -- the string to be copied
 * @return a pointer to the new copy
 * **/
char* strmcopy(char* src) {
    char* newstr = malloc((strlen(src)+1)*sizeof(char));
    strcpy(newstr, src);
    return newstr;
}
```

Weather

- Chose to malloc the space for weather here
- so I will free it all here too

file: wweather.h

```
#define MAIN_ARRAY 1
typedef struct {
    Time * time;
    Temperature * temperature;
    Temperature * dewPoint;
    int relHum;
    Wind * wind;
} WeatherData;
extern WeatherData **  weather;
void wprinter(WeatherData *w);
int readFile(char *fileName);
void freeAllWeather();
```

```
int wcount = 0; // PRIVATE VARIABLE!!!

void wprinter(WeatherData* w) { //unchanged
}
WeatherData* parse(char* line) {  //PRIVATE METHOD
    WeatherData *ret = malloc(sizeof(WeatherData));
    char *c = strtok(line, " \t");
    char *c2 = strtok(NULL, " \t");
    ret->time = makeTime(c, c2);
    c = strtok(NULL, " \t");
    c2 = strtok(NULL, " \t");
    ret->temperature = makeTemperature(atoi(c), c2);
    c = strtok(NULL, " \t");
    c2 = strtok(NULL, " \t");
    ret->dewPoint = makeTemperature(atoi(c), c2);
    c = strtok(NULL, " \t");
    ret->relHum = atoi(c);
    c = strtok(NULL, " \t");
    c2 = strtok(NULL, " \t");
    char *c3 = strtok(NULL, "\t");
    ret->wind = makeWind(c, atoi(c2), c3);
    return ret;
```

More Weather

- First step – allocate space for array of POINTERs to weather objects
  - not the objects themselves
- Note use of conditional compilation!!!
  - if MAIN_ARRAY is defined, use array notation for working with the weather array.
  - Else do it with pointers

```
int readFile(char* fileName) {
    weather = malloc(200 * sizeof(WeatherData *));
    char line[256];
    FILE *f = fopen(fileName, "r");
    if (f==NULL) {
        fprintf(stderr, "Could not open %s -- quitting\n", fileN
        return -1;
    }
    #ifndef MAIN_ARRAY
    WeatherData **cWeather = weather;
    #endif
    wcount = 0;
    while (NULL != fgets(line, 256, f)) {
        if (strlen(line)>0) {
            #ifdef MAIN_ARRAY
            weather[wcount] = parse(line);
            #else
            *cWeather = parse(line);
            cWeather++;
            #endif
            wcount++;
        }}
    fclose(f);
    return wcount;
}
```

Cleaning up weather

- freeAllWeather is public
  - freeing order is important.
  - Always free everything within a [struct or array] before freeing the thing itself!!!
- Use the destructors you defined.
- VERY java-like

```
void freeWeather(WeatherData * ww) {
    freeTime(ww->time);
    freeTemperature(ww->temperature);
    freeTemperature(ww->dewPoint);
    freeWind(ww->wind);
    free(ww);
}

void freeAllWeather() {
   for (int i = 0; i < wcount; i++) {
        freeWeather(weather[i]);
    }
    free(weather);
}
```

Lab from last week (not assigned)

- Create a struct that defines students at Bryn Mawr (very briefly).
  - The struct must have at least 2 “strings” and two integers.
    - The integers should be stored in the struct as integers (not pointers to integers).
    - The strings should be dynamically allocated at runtime to contain as little space as possible.
  - Write a constructor and destructor for this struct.
  - You may not use the strmcpy function from class today.

Faculty

first allocate space for structure

See full code for handling malloc failures

then space for sub-sections

DO NOT allocate space for base types

Alternate form for which -> is a shortcut

first free sub-sections

then free structure

```
typedef struct {
    char *firstName;
    char *lastName;
    char *department;
    int birthYear;
    int hireYear;
} Faculty;
Faculty* makeFaculty(char* fn, char* ln, char* dep, int
    Faculty *ret = malloc(1 * sizeof(Faculty));
    ret->firstName = malloc((strlen(fn)+1) * sizeof(cha
    strcpy(ret->firstName, fn);
    ret->lastName = malloc((strlen(ln)+1) * sizeof(char
    strcpy(ret->lastName, ln);
    ret->department = malloc((strlen(dep)+1) * sizeof(c
    ret->birthYear = by;
    (*ret).hireYear = hy;
    return ret;
}
void destroyFaculty(Faculty* fac) {
    free(fac->firstName);
    free(fac->lastName);
    free(fac->department);
    free(fac);
}
```

Using Faculty

print into string, then return string

semi-stupidly complex way to print, but very Java

all good? Ask valgrind!!!

```
char* Faculty2String(int strlen, char* string, Faculty * fac)
    snprintf(string, strlen, "%s %d", fac->firstName, fac->hi
    return string;
}

void printFaculty(Faculty* f) {
    char ss[200];
    printf("%s\n", Faculty2String(200, ss, f));
}


int main(int argc, char const *argv[])
{
    Faculty *f = makeFaculty("Geoff", "Towell", "CS", 1961, 20
    printFaculty(f);
    f = makeFaculty("Deepak", "Kumar", "CS", 1961, 1992);
    printFaculty(f);
    destroyFaculty(f);
    return 0;
}
```

Linked Lists

- needs a “self-referential” struct
- Can not do this with typedef as name does not exist until typedef complete.
- But can use combination of typedef and struct naming.
- constructor is straightforward
- don’t forget room for \0 in strings

```
typedef struct DLLItem {
    char *payload;
    struct DLLItem *next;
    struct DLLItem *prev;
} DLLItem;

DLLItem* makeDLLItem(char* data) {
    DLLItem *dlli = malloc(1 * sizeof(DLLItem))
    dlli->prev = NULL;
    dlli->next = NULL;
    dlli->payload = malloc((strlen(data)+1) * s
    strcpy(dlli->payload, data);
    return dlli;
}
```

Linked Lists, p2

- Suggestion: create a wrapper struct to hold info about a particular LinkedList
- Not strictly required, but certainly useful
  - technically this can be said about every struct

```
typedef struct {
    int count;
    DLLItem *head;
    DLLItem *tail;
} DLL;

DLL* makeDLL() {
    DLL *ret = malloc(1 * sizeof(DLL));
    ret->head = NULL;
    ret->tail = NULL;
    ret->count = 0;
    return ret;
}
```

Using the DLL

- Add head does exactly what you expect
- Print again as expected

```
void addDLLHead(DLL *dllC, char* data) {
    DLLItem *item = makeDLLItem(data);
    if (dllC->head == NULL)
    {
        dllC->head = item;
        dllC->tail = item;
        dllC->count = 1;
        return;
    }
    dllC->head->prev = item;
    item->next = dllC->head;
    dllC->head = item;
    dllC->count++;
}

void printDLL(DLL *dll) {
    DLLItem *item = dll->head;
    while (item!=NULL) {
        printf("%s\n", item->payload);
        item = item->next;
    }
}
```

Freeing DLLs

- Mostly standard stuff
- but a seg fault!!
  - recompile
    - gcc -g
  - valgrind a.out

```
void freeDLLItem(DLLItem *dlli) {
    free(dlli->payload);
    free(dlli);
}
void freeDLL(DLL * dll) {
    DLLItem *item = dll->head;
    while (item!=NULL) {
        freeDLLItem(item);
        item = item->next;
    }
    free(dll);
}
```

Lab

- Write remove from head for DLL
- Why is this function wrong (in the sense that it will seg fault) and how would you fix it?
-

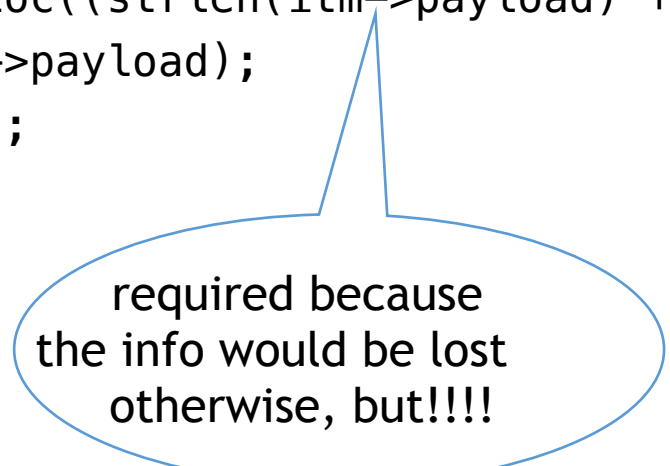
```
void freeDLL(DLL * dll) {
    DLLItem *item = dll->head;
    while (item!=NULL) {
        freeDLLItem(item);
        item = item->next;
    }
    free(dll);
}
```

Queues

- Use the DLL
  - needs more
    - removeTail
- Why not just use circular array?

- Revisit DLLItem constructor/destructor and eliminate the copy into new memory. Just take the thing supplied
- Otherwise need to take care to free the returned thing!!

```
char* removeTail(DLL *dll) {
    if (dll->count<=0)
        return NULL;
    dll->count--;
    DLLItem *itm = dll->tail;
    DLLItem *tprev = itm->prev;
    if (tprev==NULL) {
        dll->head = NULL;
        dll->tail = NULL;
        return;
    }
    dll->tail = tprev;
    tprev->next = NULL;
    char *rtn = malloc((strlen(itm->payload) + 1) * sizeof(char
    strcpy(rtn, itm->payload);
    freeDLLItem(itm);
}
```



Q Basics

- Constructor, destructor, and struct are pretty minimal

```
typedef struct {
    DLL *internal;
} Queue;

Queue* makeQueue() {
    Queue *rtn = malloc(1 * sizeof(Queue));
    rtn->internal = makeDLL();
    return rtn;
}

void freeQueue(Queue* q) {
    freeDLL(q->internal);
    free(q);
}
```

Q more

- rest is pretty basic also
- Essentially all work done by DLL!

```
void add2Queue(Queue* q, char* item) {
    addDLLHead(q->internal, item);
}

char* pullFromQueue(Queue* q) {
    return removeTail(q->internal);
}
```

Splitting & Making

- I made a single dll.c and dll.h
  - IMHO DLLItem is more a private inner class and so it does not get its own file(s)
- Also a .c and .h for queue
- Only tricky bit came when I wanted to let both dll.c and queue.c have main functions
  - like Java
- Problem c has only a single namespace so although the 2 mains cannot see each other, there are there
  - To get this I wrapped main in dll.c with #ifdef DOTO … #endif
  - In makefile added -D DOTO=1 to dll.o compile

```
CFLAGS = -g -O2

dll: dll.c dll.h
    gcc $(CFLAGS) -o dll dll.c

queue: dll.o queue.c
    gcc $(CFLAGS) -o queue dll.o queue.c

dll.o: dll.c dll.h
    gcc $(CFLAGS) -c -D DOTO=1 dll.c

clean:
    rm *.o dll queue
```