

# Lecture Notes for CIS 213, Chapter 1

Charles R. Hardnett

Spring 2004

## 1 Introduction

First day administrivia:

- Get Name, Student Number, Email address, and Phone number
- TA: TBD
- Make sure you read the syllabus. You're expected to keep up with the readings. If you're ever unsure what to read, *ask*.
- Class web page:

*<http://www.spelman.edu/compsci/cis213/>*

- Online grades will be done via webct
- Bulletin Board will be on webct
- lectures, labs, assignments will be on the class webpage
- Research Opportunities in the Department.
- Lab today: a learning diagnostic test and unix lab part I.
- ACM student chapter. A good way to get involved in the life of the department. I encourage it.
- ACM Programming Team: for those who want to spend an hour or so every week improving their programming skills, and maybe compete in a contest. Please consider joining—it's fun and educational.

## 2 Teaching Philosophy

Sometimes I'm the coach, and sometimes I'm the judge. Ultimately, success or failure is up to you. I'll help all I can, but grades are earned, not given.

This course can be difficult, and you should expect to work hard.

There is a temptation to get bogged down, but we have to keep up the pace. I can't decide to teach less in this course: if we don't cover something, eventually it'll catch up to you.

### 3 What This Course is About

Three fundamental goals:

- Algorithms and data structures: lists, trees, graphs. Their time and space requirements, called *complexity*. When to choose which.
- models/proofs: induction, finite state machines, formal languages
- breadth: getting a chance to know more about the different things that can be done in CS.

Being Successful:

- Note taking will rotate, so that everyone is not being scribes in the class.
- Reading the textbook: when to skim, when to read deeply, and when to solve problems
- study groups
- solve problems, write code fragments!

### 4 C vs C++

Everything in C is included in C++, but some things are *deprecated*—not to be used because they’re replaced by something better. The programs in your book work in C++, but may look a little odd to you. Be courageous!

- a `struct` is just like a `class`, except (1) everything is public and (2) you don’t mention the member functions.
- In C, you have to declare all your variables before your code starts; you can’t do it as you go.
- C doesn’t have constants, so you see stuff like this:

```
\#define TRUE 1
```

By convention, C programmers use all uppercase for constants. Most C++ programmers do too.

- C programmers use `typedef` more than C++ programmers

```
typedef float Money;
typedef struct StuInfo Student;
struct StuInfo {
    char* name;
    int age;
    float gpa;
}
typedef struct Node* ListNode;
```

- I/O looks different and requires a different #include: #include <stdio.h>

```

getchar()      inputs a single character
putchar()      outputs a single character
printf("%d",n) formatted output
scanf("%d",n)  formatted input; like printf in reverse

```

Printf is actually really cool. Suppose you want to output two ints and two floats, with varying numbers of digits. Here's how:

```
printf("%5d %5.2f %3d %7.4f\n",a,b,c,d);
```

- Memory allocation is different. You get a chunk of memory of the correct size, then convert it to the type you want.

```

typedef struct CELL *LIST;
struct CELL {
    int element;
    LIST next;
}

void foo() {
    LIST pCell;
    pCell = (LIST) malloc(sizeof(struct CELL));
    free(pCell);
}

```

## 5 Recap of Pointers (using C)

Pointers are used very often in the the data structures we will be creating because pointers allow the data structure to take on any formation. Also, pointers allow the data structure to be dynamic, increasing and decreasing at run-time.

```

struct StuInfo {
char* name;
int age;
float gpa;
}

struct StuInfo aStudent;

aStudent.age = 18;
aStudent.name = (char*) malloc(50);
aStudent.name = 'Erin Smith'

int x, y, *p;

p = &x // What does this do? Draw a picture?

y = *p; // picture?

x = p; // whats in x now?

```

whats another way to write `p = &x, y = *p` without pointers?

## 6 Importance of Visualization

When you are given code that describes the manipulation of a data structure, then it is important that you can visualize that data structure from the specification in the code.

```
typedef int mytype[5];

typedef mytype* mytype_ptr;

typedef struct {
int what;
mytype_ptr where;
} anotha_type;

typedef anotha_type array_type[5];

typedef struct box BOX;
struct box {
int value;
char* name;
BOX* link;
};

typedef struct box2 BOX2;
struct box2 {
char* name;
int field;
int field;
BOX2* link;
BOX2* link;
};
```

NOTE: these are all types, there are no vars here