Welcome!

COMP1511 18s1

Programming Fundamentals

COMP1511 18s1

— Lecture 14 —

Pointers + Structs + malloc

Andrew Bennett

<andrew.bennett@unsw.edu.au>

Before we begin...

introduce yourself to the person sitting next to you

how are they going with **assignment 2**?

Overview

after this lecture, you should be able to…

make progress on **assignment 2**

have a better understanding of **pointers**:
what pointers are
how to use pointers
why we use pointers

have a better understanding of **structs**

have a better understanding of **memory** in C:
dynamic memory allocation using **malloc**
the difference between

(**note**: you shouldn't be able to do all of these immediately after watching this lecture. however, this lecture should (hopefully!) give you the foundations you need to develop these skills. remember: programming is like learning any other language, it takes consistent and regular practice.)

Admin

Don’t panic!

assignment 2

(if you haven’t started yet, start **ASAP**

deadline extended to **Sunday 13th May**

assignment 1

tutor marking/feedback in progress

week 8 weekly test due tomorrow

don’t be scared!

don’t forget about **help sessions**!

see course website for details

let’s talk about **pointers**

Pointers?

before we talk about pointers, let’s take a step back…

Variables

think all the way back to week 1….

```
int age = 16;
```

what does this actually *mean*?

Variables and Functions

```c
int main(void) {
    int age = 16;
    int height = 185;
}
```

Variables and Functions and Arrays

```c
#define SIZE 5

int main(void) {
    int age = 16;
    int array[SIZE];
    foo(array);
}

void foo(int array[SIZE]) {
    int num = 10;
    array[0] = 100;
}
```

Variables and Functions and Arrays and Pointers

```c
#define SIZE 5

int main(void) {
    int age = 16;
    int array[SIZE];
    foo(array, &age);
}

void foo(int array[SIZE], int *age) {
    int num = 10;
    array[0] = 100;
    *age = 21;
}
```

re-visiting: **structs**

Arrays

arrays are a collection of many of the **same type** of variable

```
int array[10];
```

```
// ten boxes that can each hold 1 int
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
```

```
// ten boxes that can each hold 1 int
[0][1][2][3][4][5][6][7][8][9]
```

Structs

structs are a collection of many of **different types** of variables

```
struct student {
    int zid;
    char name[MAX_NAME_LEN];
    int ass1_mark;
};
```

```
// one box that can hold an int
[5112345]
// MAX_NAME_LEN boxes that can hold a char
[A][n][d][r][e][w][\0][ ][ ]
// one box that can hold an int
[94.5]
```

Structs

structs are a collection of many of **different types** of variables

```
struct student {
    int zid;
    char name[MAX_NAME_LEN];
    int ass1_mark;
};
```

```
struct student andrew;
andrew.zid = 5112345;
andrew.ass1_mark = 94.5;
strcpy(andrew.name, "Andrew");
```

```
// one box that can hold an int
[5112345]
// MAX_NAME_LEN boxes that can hold a char
[A][n][d][r][e][w][\0][ ][ ]
// one box that can hold an int
[94.5]
```

Arrays of Structs?

```
struct student {
    int zid;
    char name[MAX_NAME_LEN];
    int ass1_mark;
};
```

```
struct student students[NUM_STUDENTS];
// fill out one student struct in the array of structs
students[0].zid = 5112345;
students[0].ass1_mark = 94.5;
strcpy(students[0].name, "Andrew");

// fill out another student struct in the array of structs
students[1].zid = 9100123;
students[2].ass1_mark = 64.2;
strcpy(students[3].name, "Andrew");
```

let’s play: **Intensity**

Intensity

your task: write a program to **play** the game *Intensity*

the *Intensity* **referee** manages the game

shuffles cards

deals cards

asks players for moves

etc

all input is given over **standard input**

(i.e. scanf)

all output is given over **standard output**

(i.e. printf)

Stateless AI

an important concept to understand: your AI is **stateless**

it comes to life for **one** single move
reads the input
thinks about what to do
prints out its decision

Intensity Referee

the ***Intensity*** **Referee** runs the game

```
1511 intensity_referee
```

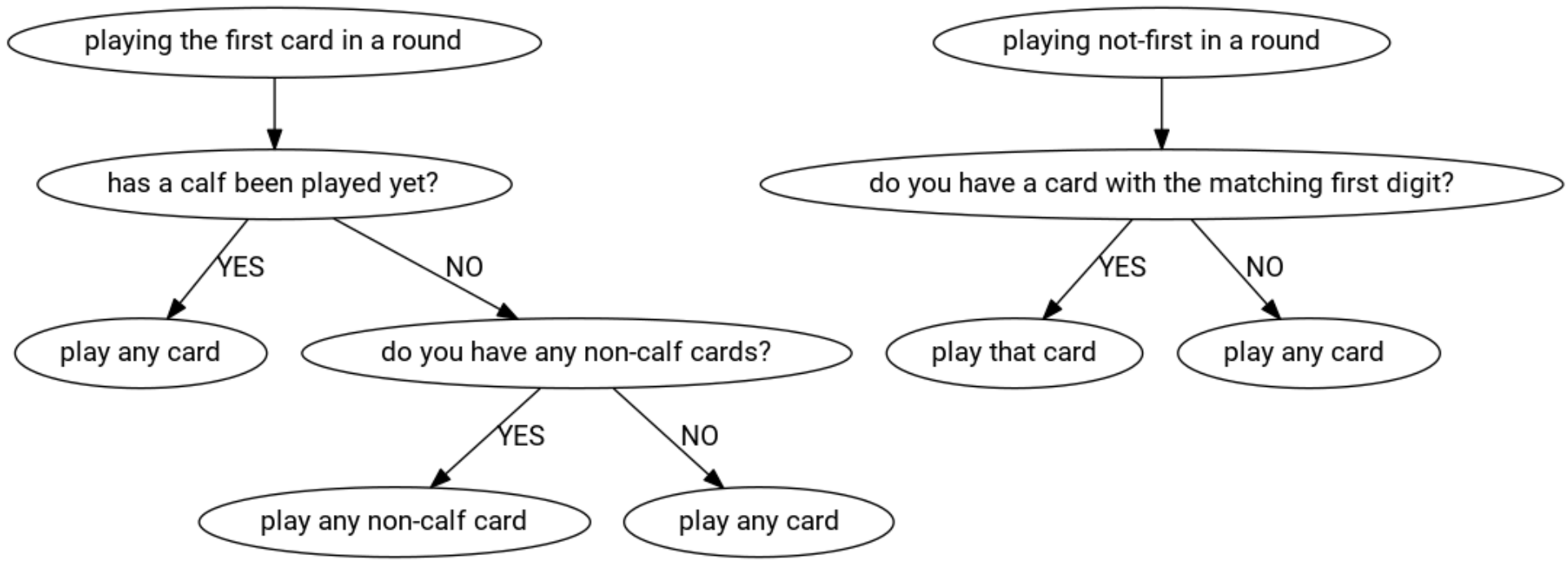
you can run your AI against it:

```
1511 intensity_referee your_ai_code.c
```

you can play interactively:

```
1511 intensity_referee -i
```

Valid Cards To Play



revisiting: memory

Scope and Lifetimes

the variables inside a function only exist as long as the function does

once your function returns, the variables inside are “gone”

(this is why you can’t return an array from a function!)

Lifetimes

what if we need something to “stick around” for longer?

two options:
make it in a “parent” function
dynamically allocate memory

Lifetimes

make it in a “parent” function

```
void foo(void) {
    int array[SIZE];
    bar(array);
    printf("%d", array[0]);
}

void bar(int array[SIZE]) {
    array[0] = 123;
}
```

Lifetimes

dynamically allocate memory

```
void foo(void) {
    int *array = bar();
    printf("%d", array[0]);
}

int *bar(void) {
    int *array = malloc(SIZE * sizeof(int));
    array[0] = 123;
    return array;
}
```