CS1010 Tutorial 2

Agenda for Today

- Problem Set 3
- Problem Set 5
- Exercise 0
- Practical: Exercise 1

Problem 3.1 – Getting MAD

- Using functions we have seen, and given a list L with k elements, find

$$\frac{\sum_{i=0}^{k-1} |l_i - \mu|}{k}$$

- Normally,
  - Math expressions evaluated “inside out”
  - What operations should you do?
  - What order should you do them in?
  - **Think computationally**
    - Not operating in abstract math sense, but with a real “list of elements”

Problem 3.1 – Getting MAD

- Using functions we have seen, and given a list L with k elements, find

$$\frac{\sum_{i=0}^{k-1} |l_i - \mu|}{k}$$

- $range(L, k)$ - returns **range** of a list of k integers
- $mean(L, k)$ - returns the **mean** of a list of k integers
- $subtract(L, k, \mu)$ - **subtracts** μ from each element in a list of k integers
- $square(L, k)$ - **squares** each element in a list of k integers
- $sum(L, k)$ - returns the **sum** of all elements in a list of k integers

- Do we need a new function?

Problem 3.1 – Getting MAD

- Using functions we have seen, and given a list L with k elements, find

$$\frac{\sum_{i=0}^{k-1} |l_i - \mu|}{k}$$

- We need a new function that replaces every element in a list of integers with its *absolute value (modulus)*
- By wishful thinking, assume that we have such a function, call it $abs(L, k)$.

Problem 3.1 – Getting MAD

- Using functions we have seen, and given a list L with k elements, find

$$\frac{\sum_{i=0}^{k-1} |l_i - \mu|}{k}$$

- We return:

$$\frac{sum\big(abs\big(subtract\big(L, mean(L,k)\big),k\big),k\big)}{k}$$

Problem 3.1 – Getting MAD

- Using functions we have seen, and given a list L with k elements, find

$$\frac{\sum_{i=0}^{k-1} |l_i - \mu|}{k}$$

- We can also return

$$mean\big(abs\big(subtract\big(L, mean(L, k)\big), k\big), k\big)$$

Problem 3.1 – Getting MAD

- Using functions we have seen, and given a list L with k elements, find

$$\frac{\sum_{i=0}^{k-1} |l_i - \mu|}{k}$$

- If a function can "know" the length of a list without the input k, we can simplify further to

$$mean\left(abs\left(subtract\left(L, mean(L)\right)\right)\right)$$

Problem 3.2(a)

- Give a **recursive** algorithm to find the sum of a list L with k elements

- Hint: take inspiration from max' and $factorial$

Problem 3.2(a)

- Give a **recursive** algorithm to find the sum of a list L with k elements

- Start with the definition
  - Let $sum(L, i, j)$ be a function that calculates the sum from elements L_i to L_j

- What is the trivial (base) case?
  - A list of one element, which means $i = j$
  - When $i = j$
    - $sum(L, i, j) = L[i]$

Problem 3.2(a)

- Give a **recursive** algorithm to find the sum of a list L with k elements

- What if there is more than 1 element?
  - Then $i \neq j$

- Then
  - $sum(L, i, j) = L_i + sum(L, i + 1, j)$

- Note that we "call" this function with $i = 0$ and $j = k - 1$
  - i.e $sum(L, 0, k - 1)$

Problem 3.2(a)

- Give a **recursive** algorithm to find the sum of a list L with k elements
  - **Draw on board**

Problem 3.2(a) EXTRA

- Give a **recursive** algorithm to find the sum of a list L with k elements by **dividing the list into two halves** and recursively finding the sum in the left half and right half
  - **Draw on board**

Problem 3.2(b)

- Give a **recursive** algorithm to compute i^j
- Like finding the sum
- What is the trivial case?
  - $i^0 = 1$ for any i

Problem 3.2(b)

- Give a **recursive** algorithm to compute i^j

- What if $j \neq 0$?
  - If $j < 0$?
  - If $j > 0$?

- **Draw on the board…**

Problem 3.2(b) EXTRA

- Give a **recursive** algorithm to compute i^j **efficiently**

- How many multiplications are done?
  - Roughly j number of multiplications
  - Twice if j is negative
    - $\frac{1}{a}$ requires one multiplication by itself

- $O(j)$ number of multiplications

- Multiplications are generally **expensive** even on modern hardware

Problem 3.2(b) EXTRA

- Give a **recursive** algorithm to compute i^j **efficiently**

- There is a (very) fast exponentiation algorithm
  - Exponentiation by squaring

- $$i^j = \begin{cases} i(i^2)^{\frac{j-1}{2}}, & \text{if n is odd} \\ (i^2)^{\frac{j}{2}}, & \text{if n is even} \end{cases}$$

- $O(\log n)$ number of multiplications

Recursive Problem Solving

- To create solutions to such recursive problems
- Identify the **base case**
  - A case so trivial it can be solved nearly immediately
- Figure out how to use solutions to **smaller problem sizes** to solve **larger problem sizes**
  - Use the trivial case to solve a case of size 2
  - Then use a case of size 2 to solve a case of size 3
  - Etc etc
- However, this is not enough for some recursive problems…
  - We'll see as the semester passes

Problem 5.1 – Become a C compiler

- Given a bunch of program examples, use a C compiler to check for compilation errors.
- I've used **clang**, but **gcc** also works
- What can you infer about what is allowed or not allowed in C?
- What could go wrong even if C does not throw compiler warnings/errors?

Problem 5.1 – Become a C compiler

- Purpose of exercise
- Common pitfalls when programming in C
- Learn to read error messages

Problem 5.1(a)

```
1  #include <math.h>
2  long square(long x)
3  {
4    return x * x;
5  }
6
7  double hypotenuse_of(long base, long height)
8  {
9    return sqrt(square(base) + square(height));
10 }
11
12 int main()
13 {
14   hypotenuse_of(3.0, 4.0); // <- use 3.0 and 4.0 instead of int.
15 }
```

Problem 5.1(b)

```
1  #include <math.h>
2  long square(long x)
3  {
4    return x * x;
5  }
6
7  double hypotenuse_of(long base, long height)
8  {
9    return sqrt(square(base) + square(height));
10 }
11
12 int main()
13 {
14   long h = hypotenuse_of(3, 4); // <-- assign to a long variable
15 }
```

Problem 5.1(c)

```
1  #include <math.h>
2  long square(long x)
3  {
4    return x * x;
5  }
6
7  double hypotenuse_of(long base, long height)
8  {
9    return sqrt(square(base) + square(height));
10 }
11
12 int main()
13 {
14   Hypotenuse_Of(3, 4); // <-- use a different case
15 }
```

Problem 5.1(c) – Compiler

```
$ clang 5.1c.c -lm
5.1c.c:14:9: warning: implicit declaration of function 'Hypotenuse_Of' is invalid in C99 [-Wimplicit-function-declaration]
        Hypotenuse_Of(3, 4); // <-- use a different case
        ^
1 warning generated.
/usr/bin/ld: /tmp/5-76f1b6.o: in function `main':
5.1c.c:(.text+0x75): undefined reference to `Hypotenuse_Of'
clang: error: linker command failed with exit code 1 (use -v to see invocation)
$
```

Problem 5.1(d)

```
1  #include <math.h>
2  long square(long x)
3  {
4    return x * x;
5  }
6
7  double hypotenuse_of(long base, long height)
8  {
9    long base; // <-- declare a new base
10   return sqrt(square(base) + square(height));
11 }
12
13 int main()
14 {
15   hypotenuse_of(3, 4);
16 }
```

Problem 5.1(d) – Compiler

```
$ clang 5.1d.c
5.1d.c:9:14: error: redefinition of 'base'
        long base; // <-- declare a new base
             ^
5.1d.c:7:27: note: previous definition is here
double hypotenuse_of(long base, long height)
                          ^
1 error generated.
$
```

Problem 5.1(e)

```
1  #include <math.h>
2  long square(long x)
3  {
4    long sqr = x * x; // <-- declare and assign in one go
5    return sqr;
6  }
7
8  double hypotenuse_of(long base, long height)
9  {
10   return sqrt(square(base) + square(height));
11 }
12
13 int main()
14 {
15   hypotenuse_of(3, 4);
16 }
```

Problem 5.1(f)

```
1  #include <math.h>
2
3  long sqr;  // <-- use global variable
4  long square(long x)
5  {
6    sqr = x * x;
7    return sqr;
8  }
9
10 double hypotenuse_of(long base, long height)
11 {
12   return sqrt(square(base) + square(height));
13 }
14
15 int main()
16 {
17   hypotenuse_of(3, 4);
18 }
```

Problem 5.1(g)

```
1  #include <math.h>
2
3  int main()
4  {
5    long square(long x) // <-- define function within function.
6    {
7      return x * x;
8    }
9
10   double hypotenuse_of(long base, long height) // <--
11   {
12     return sqrt(square(base) + square(height));
13   }
14
15   hypotenuse_of(3, 4);
16 }
```

Problem 5.1(g) – Compiler

```
$ clang 5.1g.c -lm
5.1g.c:6:9: error: function definition is not allowed here
        {
        ^
5.1g.c:11:9: error: function definition is not allowed here
        {
        ^
5.1g.c:15:9: warning: implicit declaration of function
'hypotenuse_of' is invalid in C99 [-Wimplicit-function-declaration]
        hypotenuse_of(3, 4);
        ^
1 warning and 2 errors generated.
$
```

Problem 5.1 Bonus

```c
#include <stdio.h>

int f(int x)
{
        int increment(int);

        return increment(x);
}

int increment(int n)
{
        return n + 1;
}

int main()
{
        printf("%d\n", f(100));
}
```

Echo

- Read an integer from the standard input, and print to the standard output

```
#include "cs1010.h"

int main()
{
    long x = cs1010_read_long();
    cs1010_println_long(x);
}
```

CS1010 I/O Library Functions

```
long    cs1010_read_long()
double  cs1010_read_double()
char*   cs1010_read_word()
char*   cs1010_read_line()

void    cs1010_println_long(long x)
void    cs1010_println_double(double x)
void    cs1010_println_string(char *string)
```

Divide

- Integer division possibly causes truncation of decimal part of digit

- E.g
  - $\frac{10}{3} = 3.333 \rightarrow 3$
  - $\frac{25}{5} = 5.000 \rightarrow 5$
  - $-\frac{10}{3} = -3.333 \rightarrow -3$

- This behaviour is enforced by the C standard

Divide

- To avoid, do **typecasting**

- `double x = (double)10 / 3`
  - `x == 3.333`
  - This is correct ☺

- What about
  - `double x = 10 / 3`

Quadratic

- **Show live example**

Exercise 1

- ssh into the PE nodes
- Run **`~cs1010/get-ex01`**
- Try it out

- A leap year is a calendar year containing an extra day to synchronize the calendar to seasons and astronomical events. In the Gregorian calendar, years that are multiples of four (except for years divisible by 100 but not by 400) are leap years.

- Complete the program leap.c so that it reads in an integer representing a year from the standard input and prints out " is a leap year" if the input is a leap year. Otherwise, print " is not a leap year" to the standard output.

- Your program should include a bool function is_leap_year that takes in the input year and returns true if the input is a leap year and returns false otherwise.

- ~~A leap year is a calendar year containing an extra day to synchronize the calendar to seasons and astronomical events. In the Gregorian calendar,~~ years that are multiples of four (except for years divisible by 100 but not by 400) are leap years.

- Complete the program leap.c so that it reads in an integer representing a year from the standard input and prints out " is a leap year" if the input is a leap year. Otherwise, print " is not a leap year" to the standard output.

- Your program should include a bool function is_leap_year that takes in the input year and returns true if the input is a leap year and returns false otherwise.

- ~~A leap year is a calendar year containing an extra day to synchronize the calendar to seasons and astronomical events. In the Gregorian calendar,~~ years that are multiples of four (except for years divisible by 100 but not by 400) are leap years.

INPUT

- Complete the program leap.c so that it reads in an integer representing a year from the standard input and prints out " is a leap year" if the input is a leap year. Otherwise, print " is not a leap year" to the standard output.

OUTPUT

- Your program should include a bool function is_leap_year that takes in the input year and returns true if the input is a leap year and returns false otherwise.

- ~~A leap year is a calendar year containing an extra day to synchronize the calendar to seasons and astronomical events. In the Gregorian calendar,~~ years that are multiples of four (except for years divisible by 100 but not by 400) are leap years.

ALGORITHM

INPUT

- Complete the program leap.c so that it reads in an integer representing a year from the standard input and prints out " is a leap year" if the input is a leap year. Otherwise, print " is not a leap year" to the standard output.

OUTPUT

CONSTRAINTS

- Your program should include a bool function is_leap_year that takes in the input year and returns true if the input is a leap year and returns false otherwise.