Variables

C Review

- Every C program needs 5 parts:

  1) Comment with your name >> */*Mr. Malloy*/*

  2) Preprocessor directive(s) >> *#include <stdio.h>*

  3) >> *int main (void)*

  4) Brackets >> *{ /*Code here */ }*

  5) Return 0 >> { /*Code here */ return 0; }

Variables

- Allow things (numbers, letters, words, etc…) to be stored and manipulated

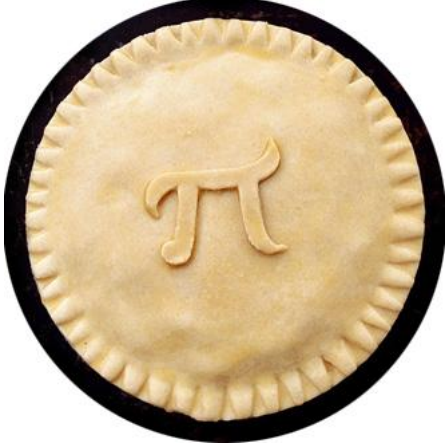
- Each variable is a SINGLE memory address

Variables

- But each computer has billions of memory addresses

- Example: A 4 GB phone has 4,000,000,000 bytes
  - 1 byte = 8 bits, so 32,000,000,000 memory addresses

Variables



- Therefore, we cannot use variables in our code - it becomes too unwieldy

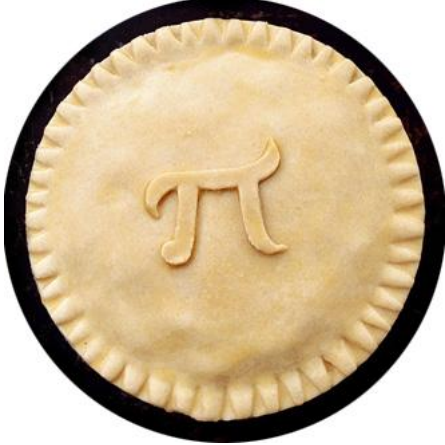
0xABCD1200 = 3.14159;

0xABCD1204 = 6;

*0xABCD1208 = 0xABCD1204 * 0xABCD1204;*

*0XABCD120C = 0xABCD1200 * 0xABCD1208;*

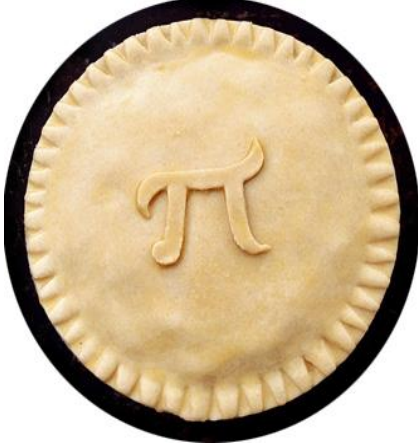
Variables



- C (and nearly all programming languages) gives names to variables to prevent this problem

```
PI = 3.14159;

radius= 6;

radius_squared = radius * radius ;

area = PI * radius_squared;
```

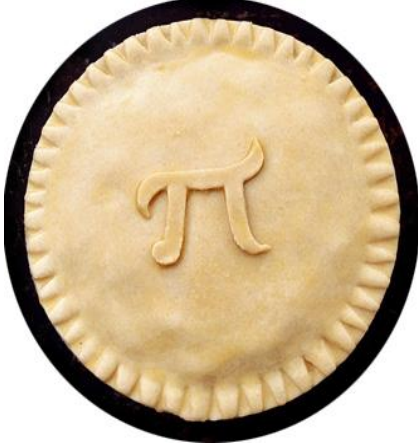
Variables



- Similar to math variables in that assignment is from left to right

```
PI = 3.14159;

radius= 6;

radius_squared = radius * radius ;

area = PI * radius_squared;
```

Variables

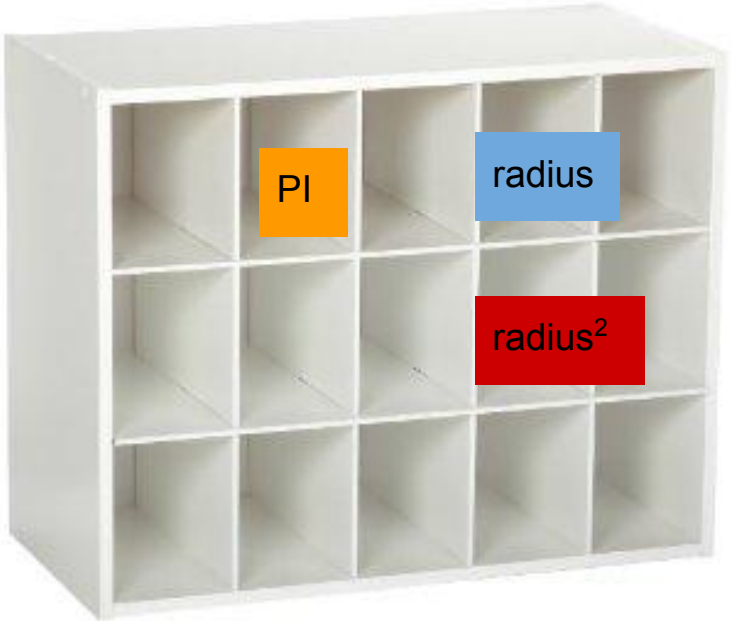


- Similar to math variables in that assignment is from left to right → the variable PI equals 3.14159

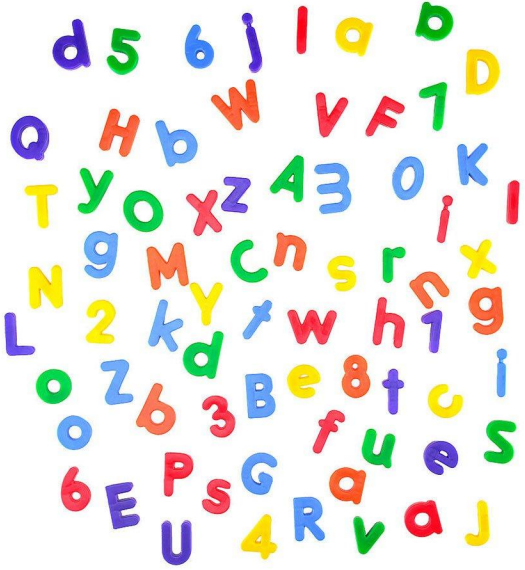
```
PI = 3.14159;

radius= 6;

radius_squared = radius * radius ;

area = PI * radius_squared;
```

Variable Addresses

- In C, each variable refers to a DIFFERENT memory address
  - This is unique to C - other languages have different rules



Variable Names

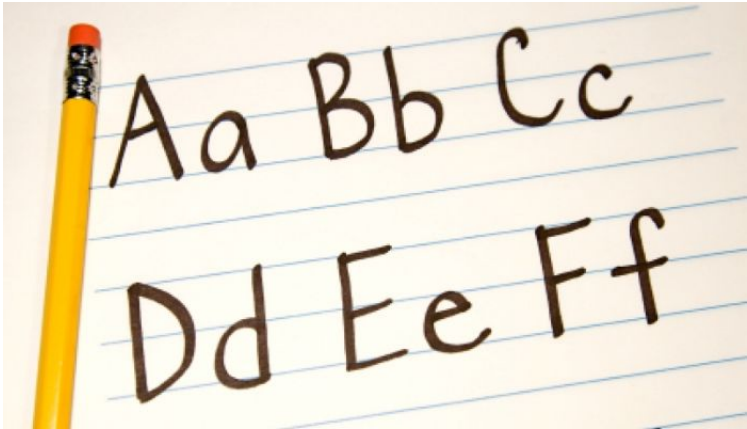


- Uniquely identify a variable in C
- Can include: letters, numbers, and underscores
  - C won't allow anything else!

- >> *Pie = good*
- >> Pie45_morePie = *good*
- >> Pie? = *bad*

Variable names

- First letter HAS to be a letter or an underscores
  - >> _pie4 = *good*
  - >> 4_pie = *bad*
- Variable names are case sensitive
  - Similar to file names
  - **Pie** *is not the same as* **pie**

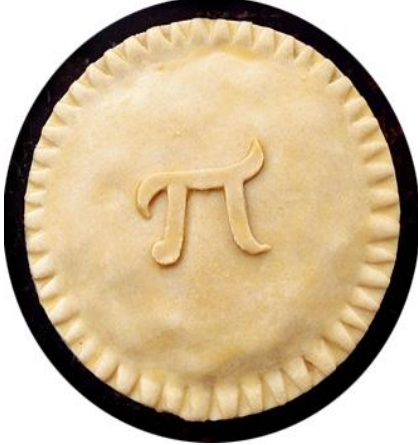


Variable names

- Cannot have the same name as *keywords*
  - Keywords in C are protected - are used for different things

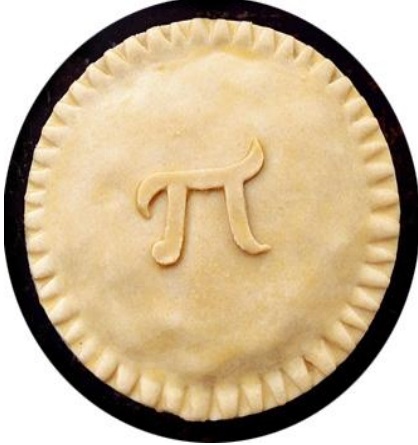
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Variable names



- C Conventions
  - Traditional way to name variables in C
    - All lowercase
    - Words separated by underscores
  - >> pie_pi
  - >> awesome_pi

Variable names



- C Conventions
  - Constants (variables whose value will not change) are in ALL CAPS
    - >> PI = 3.14159
    - >> LENGTH_OF_CLASS = 64
    - >> GRAVITY = 9.8
  - Iterators (used in repetitions) are a single letter
    - i, j, k...

Variable declaration

- C has to know what KIND of variable you are using
  - Number
  - Letter
  - True/False (boolean)
  - Etc…

Variable declaration

- C has to know what KIND of variable you are using

- Variable types (3 out of many)
  - int → number (no decimal)
  - float → number (decimal)
  - bool → 1 or 0 (true or false)

Variable declaration

- Declare a variable by writing the *variable type*, then the *variable name*

>> int score;

>> float PI;

>> bool present_in_school;

Variable declaration

- Can declare multiple variables in one line (as long as they are the same type)

>> int ravens_score, jags_score, total_score;

- Makes three variables, all integers, named:
  - ravens_score
  - jags_score
  - total_score

Variable declaration - behind the scenes

- The compiler is allocating a specific space in memory for the variable
  - >> int ravens_score →

Declaration vs Initialization

- Declaration makes the variable exist
  - The compiler allocates space, but doesn't "fill" it with anything specific
  - >> int ravens_score, jags_score, total_score;
- Initialization assigns the variable a value
  - >> ravens_score = 19;
  - >> jags_score = 17;

Displaying a variable

- Use the *printf* function to display variables

  >> printf(“The Ravens score was %d\n”, ravens_score);

- %d → format specifier. Replaced on the screen by the variable it specifies
- %f → used for floating point values

  >> printf(“The value of pi is %f\n”, PI);

Displaying a variable

- Can display more than one variable at a time

>> printf(“The Ravens scored %d points, while the Jaguars scored %d points”, ravens_score, jags_score);

- **Have to be listed in order - if the variables are flipped at the end, then they are flipped in the print statement**

Variable standards

- Add a comment when initializing / declaring variables

  >> /*This variable is the total score of the Ravens-Jags game*/

  >> int total_score;

Variables

- Can perform mathematical operations with variables
  - >> total_score = ravens_score + jags_score;
  - >> float average_score = total_score / 2;
  - >> int ravens_score_times_three = ravens_score * 3;

Variable assignment

- Print out the average score of the winning team from the NFL games this weekend
  - Each winning score must be a unique variable
  - All math must be done using variables
  - The average score must include decimals
  - Code must compile
  - Save to Desktop - will upload to github at the end of class