24-780B
Engineering Computation

Mon/Wed 4:40-6:40pm, Fall 2022

Lecture 2

Nestor Gomez, Ph.D., P.E.

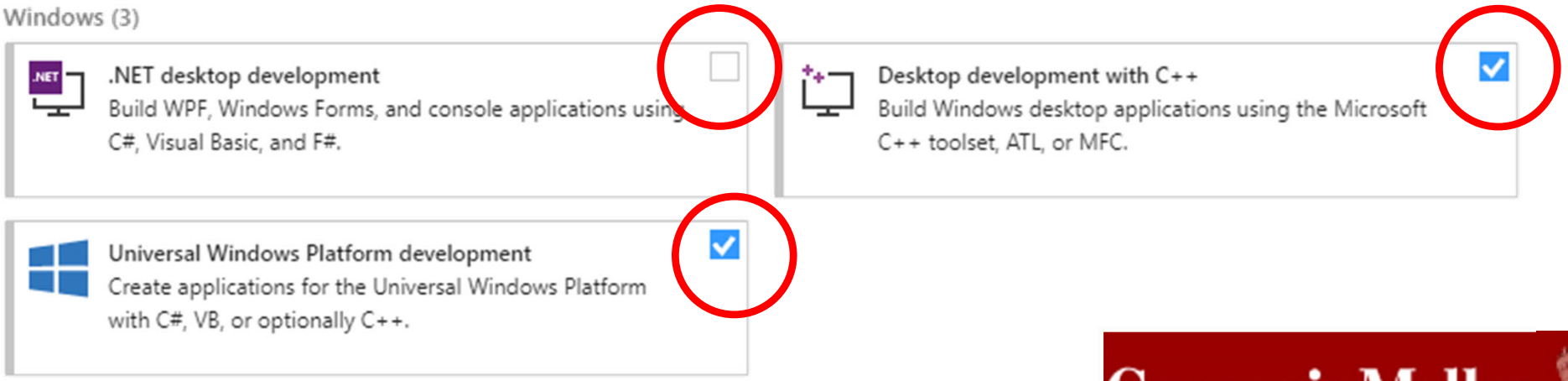
nestor@cmu.edu

Carnegie Mellon

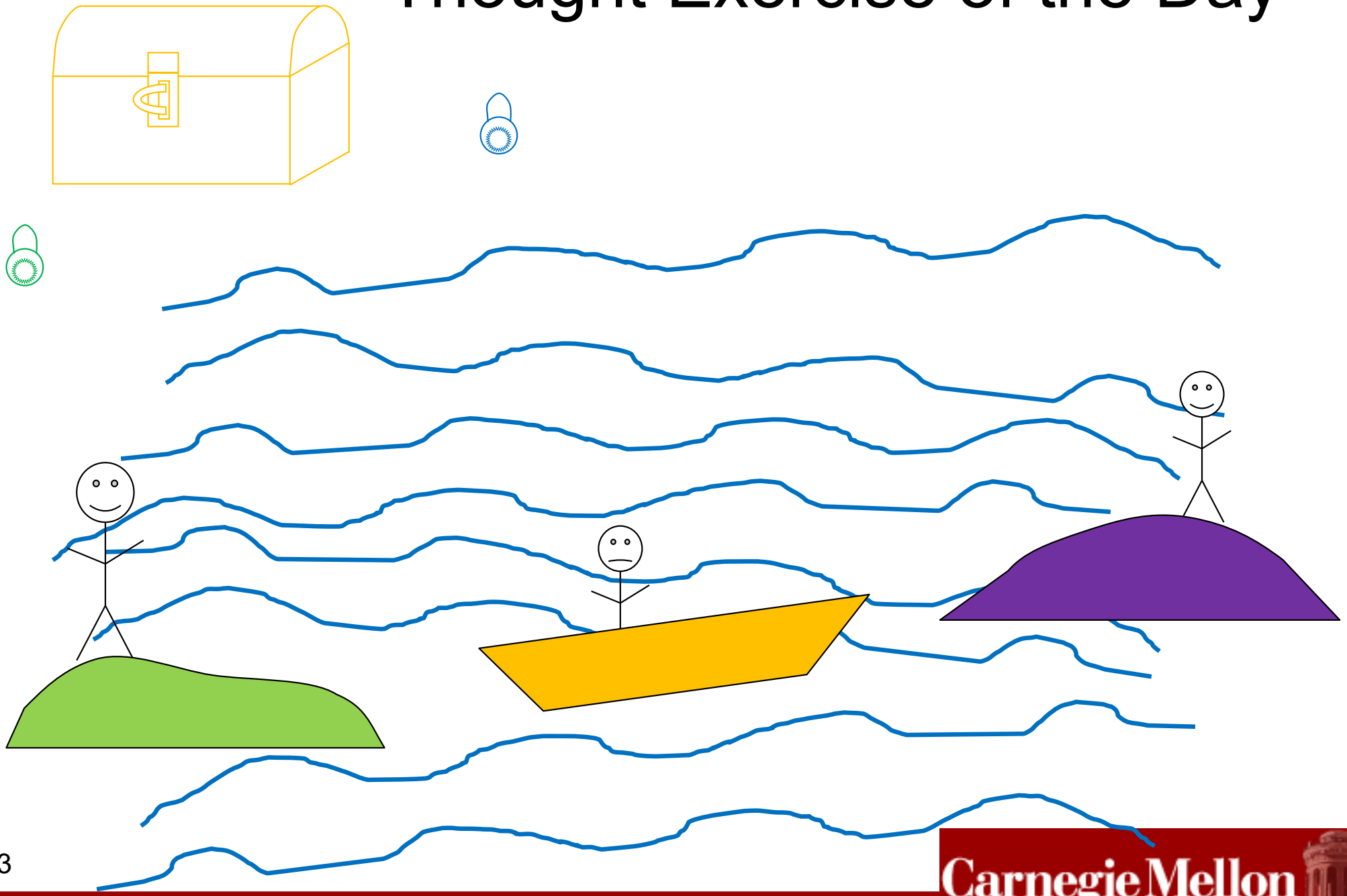
Visual Studio 2022 Community Edition

- Download and install (30 minutes?) from: https://visualstudio.microsoft.com/downloads/

- Select “Community Edition” and be sure to include “Universal Platform” and “Desktop development with C++”



Thought Exercise of the Day



Secret Tips for Becoming a Great Programmer

- ~~Start young~~
- ~~Work long, work late~~
- ~~Follow the rules~~
- ~~Read lots of books~~
- ~~Watch tons of videos~~
- ~~Absorb dozens of languages~~
- Write code, see code, understand code

Arrays

- When you need to store several related variables.

```
int a0,a1,a2,a3,a4;
```

- It is better to store them as an array

```
int a[5];
```

- Think of an array as a subscriptable variable you can access just like the original.
- Instantiate/Declare like this:

```
int a0=12, a1=34, a2=56, a3=78, a4=910;
int a[5] = { 12, 34, 56, 78, 910};
// or simply -> int a[] = {12, 34, 56, 78, 910};
```

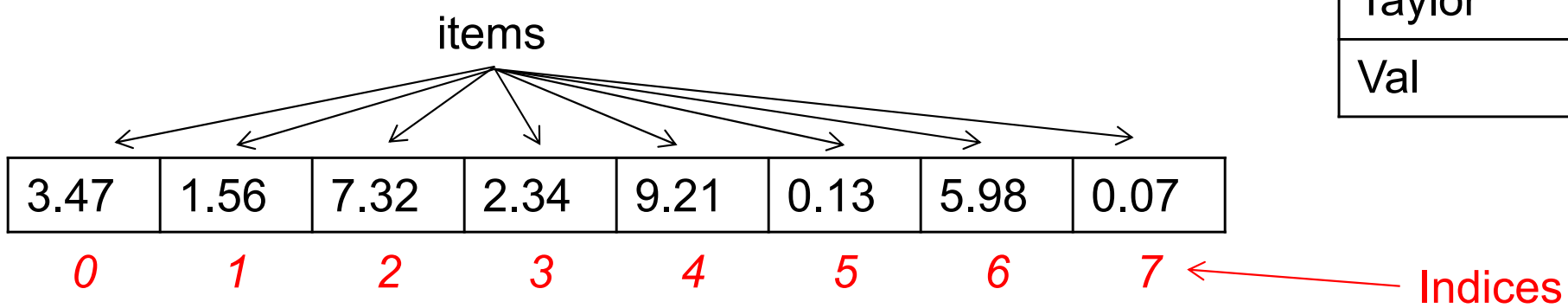
- Then use like this:

```
cout << “a3=“ << a3 << endl;
cout << “a[3]=“ << a[3] << endl;
```

Array

- An array is collection of data, all of the same type (e.g., an array of double, an array of string, etc.)
- Great because access is very fast
- Typically, it is represented as a table-looking diagram:

Pat
Drew
Sam
Alex
Taylor
Val



Arrays

Printing number of days of a month using an array.

```
#include <stdio.h>
int main(void)
{
   int daysOfMonth[] = { 31,28,31,30,31,30,31,31,30,31,30,31 };
   int month;

   printf("Month (1=January):");
   scanf("%d",&month);

   if(1<=month && month<=12)
   {
      printf("Number of days in month %d in a regular year is %d",
                 month, daysOfMonth[month-1]);
   }
   return 0;
}
```

Array

- To declare:

```
type varName[size];
type varName[] = { comma-separated list};
```

- Examples:

Why?

```
double dailyTemperature[367];
string myBestFriends[] = { "Mom", "Santa",
                           "Snuggle Bear"};
int goGrid[19][19] = { }; // sets all values to 0
```

Arrays

1	2	3	4
5	6	9	8

- Arrays can be declared for any "dimension":

```
int array2D[3][6];   // array with 3 rows, 6 columns
int table[][4] = { { 1, 2, 3, 4}, { 5, 6, 9, 8 } };
```

- For above table, what is the value of *table[1][2]*?
- Note that the number of rows can be "implied", but the number of columns cannot.
- Arrays with more than 3 dimensions are difficult to visualize, so be careful that you actually need such a thing.
- Arrays can be 1D (row or column), 2D (table/sheet), 3D (book)*, 4D (bookshelf?), 5D (bookcase?), etc.*
- Dimensionality can be used to make access clearer for programmer, but it does not change how memory is used.

The two most important program control mechanisms

- Conditional Branching
  - If-then-else
  - Switch-case
- Looping
  - For-loop
  - While-loop
  - Do-loop (perhaps more appropriately called “do-while-loop”)

If-then-else

```
if (testscore >= 90) {
    grade = 'A';
} else if (testscore >= 80) {
    grade = 'B';
} else if (testscore >= 70) {
    grade = 'C';
} else if (testscore >= 60) {
    grade = 'D';
} else {
    grade = 'R';
}
cout << "Grade = " << grade << endl;
```

Note: Since there is only one line of each *if*, the curly brackets could have been omitted.

Could also use "\n" like in good ol' C

Switch-case

```
int i = 2;
switch (i) {
    case 1: std::cout << "1";
    case 2: std::cout << "2";  //execution starts at this case label
    case 3: std::cout << "3";
    case 4:
    case 5: std::cout << "45";
            break;          // execution of other statements is terminated
    case 6: std::cout << "6";
    default: std::cout << “Nothing”;
}
```

Used for discrete types (NOT double or string)

Often used with enumerated types (more later)

For Loops

for(*initializations*; *condition which keeps you in loop*;
what to do at the end of each loop) {

statement1;

statement2;

. . .

}

Example

```
for(int i=1; i<=100; i++) {
   // if switch number i is "on" print out the number i
}
```

While Loops

```
while( boolean expression ) {
   statement1;
   statement2;
   .  .  .
}
```

Example

```
i = 1;
while(i<=100) {
   // check switch i
   // print it out
   i++;
}
```

Do-while Loops

```
do {
   statement1;
   statement2;
   .  .  .
} while( boolean expression );
```

Example

```
i = 0;
do {
   i++;
   // check switch i
   // print it out
} while(i<100);
```

How/when to use which loop?

- In C/C++, as you may have noticed, the three kinds of loops are essentially interchangeable
- For-loop
  - Good when we know number of iterations before entering loop
  - Tradition discourages changing counter anywhere but in the for-statement itself at start of loop
  - Can add “who knows what can happen” condition(s)
- While-loop
  - Best when the condition for staying or exiting loop is affected in the loop itself
- Do-while-loop
  - Just like while-loop, but the loop body is guaranteed to run at least once.

For Problem Set 1

- Start by getting Visual Studio C++ or Xcode C++ to work.
- Think about what function(s) are needed (although we have not discussed functions very much)
- Set up variables (maybe “hard-wired” while testing)
- Once it is working, maybe think about optimizing/refining

- To stop program and wait for Enter Key use `system(pause);`

Comments at top of my solution

```
/*
     3.14159265358979323846264338327 9
   502884197169399375105820974944592 3
  078164062862089986280348253421170 67
  9821    48086         5132
 823      06647        09384
 4        60955        05822        Nestor Gomez
 3        17253        5940         Carnegie Mellon University
          8128         4811         Engineering Computation, 24-780B
          1745         0284         PS01.  Due Tues. Sept. 6, 2022
          1027         0193
         85211        05559         Function for calculating the value
         64462        29489         of pi by adding discrete areas located
         5493         03819         within quarter circle.
         6442         88109
        75665         93344
       612847         56482
       33786          78316        52
      7120190         914564       85
     6692346          03486104543266
    4821339           3607260249141
    2737245            87006606315
    588174              881520920"

*/
```

High-Low Game

Let’s make a high-low game.

The user makes a guess of a random number.

(Let the user play a guessing game.)

High-Low game

Breaking down into details.

1. Computer comes up with a random number.
2. You make a guess.
3. The computer will tell you if your guess is right or wrong.

Your program needs to check if the answer is right or wrong.

High-Low game

Think about the data structure needed for the program.

Can you guess?

1. Computer comes up with a random number.
2. You make a guess.
3. The computer will tell you if your guess is right or wrong.
4. Perhaps later it will provide feedback on your guess and allow more guesses.

High-Low game

Think about the data structure needed for the program.

1. Computer comes up with a random number.
2. You make a guess.
3. The computer will tell you if your guess is right or wrong.

You may want to make the random number more random by giving a seed to the C++'s random number generator. Typically the seed is taken from the timer.

-> An additional data: Current Time.

High-Low game

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    int answer, guess, currentTime;

    currentTime=time(NULL);
    srand(currentTime);

    answer=rand();
    answer=answer%10;

    printf("Make a guess (0 to 9):");
    scanf("%d",&guess);

    if(answer==guess) {
          printf("You've got the right answer!\n");
    }
    else {
          printf("Wrong. The answer is %d\n",answer);
    }

    return 0;
}
```

Want to use Standard Input/Output, Standard Library (includes random number generator), and timer library

Make random-number generator more random

Generate a random number

Take a remainder of division by 10. Force it to be 0 to 9.

Take an integer as input

Conditional expression

High-Low game (C++ input/output)

```
#include <iostream>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    int answer, guess, currentTime;

    currentTime=time(NULL);
    srand(currentTime);

    answer=rand();
    answer=answer%10;

    std::cout >> "Make a guess (0 to 9):");
    std::cin >> guess;

    if(answer==guess) {
          printf("You've got the right answer!\n");
    }
    else {
          printf("Wrong. The answer is %d\n",answer);
    }

    return 0;
}
```

Want to use Input/Output streams, Standard Library (includes random number generator), and timer library

Make random-number generator more random

Generate a random number

Take a remainder of division by 10. Force it to be 0 to 9.

Take an integer as input

Conditional expression

High-Low game

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

This tells the compiler that you want to use the following sets of tools:

- Standard Input/Output
- Standard Library
- Timer Library
- Stream Input/Output

High-Low game

```
int answer, guess, currentTime;
```

This tells the compiler that you are going to use three variables, answer, guess, and currentTime, and all of them are integer.

```
currentTime = time(NULL);
```

Seconds since 0:00 on January 1st 1970.
(Have you ever heard about Y2038 problem?)

```
srand(currentTime);
```

This gives a seed number to the random-number generator. For the same seed number, the random-number generator will generate exactly same set of random numbers.

High-Low game

```
answer=rand();
```

This generates a random number and assigns it to ‘answer’.

```
answer=answer%10;
```

Since I want to make it from 0 to 9, take remainder of division by 10. “%” is the modulo operator in C/C++

```
printf("Make a guess (0 to 9):");
```

Prompt the user to make a guess.

```
scanf("%d",&guess);
```

Takes an integer as input from the user. %d is for integer just like printf.

High-Low game

```
if(answer==guess) {
    printf("You've got the right answer!\n");
}
else {
    printf("Wrong.  The answer is %d\n",answer);
}
```

Conditional expression. ‘==‘ is for comparison.

==	Equal
!=	Not equal
<	Less
>	Greater
<=	Less or equal
>=	Greater of equal

High-Low game

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    int answer,guess,currentTime;

    currentTime=time(NULL);
    srand(currentTime);

    answer=rand();
    answer=answer%10;

    printf("Make a guess (0 to 9):");
    scanf("%d",&guess);

    if(answer==guess) {
          printf("You've got the right answer!\n");
    }
    else {
          printf("Wrong.  The answer is %d\n",answer);
    }
    return 0;
}
```

What are these? We will come back to these when we talk about functions.

Functions

Let's revisit the High-Low game.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
    int answer,guess,currentTime;

    currentTime=time(NULL);
    srand(currentTime);

    answer=rand();
    answer=answer%10;

    printf("Make a guess (0 to 9):");
    scanf("%d",&guess);

    if(answer==guess)
    {
        printf("You've got the right answer!\n");
    }
    else
    {
        printf("Wrong.  The answer is %d\n",answer);
    }
    return 0;
}
```

Setting a seed for the random-number generator

Generating a random number (0 to 9)

Take an input from the user

Show result

Functions

Wouldn't it be nice if we could write like this? (i.e., the way we think).

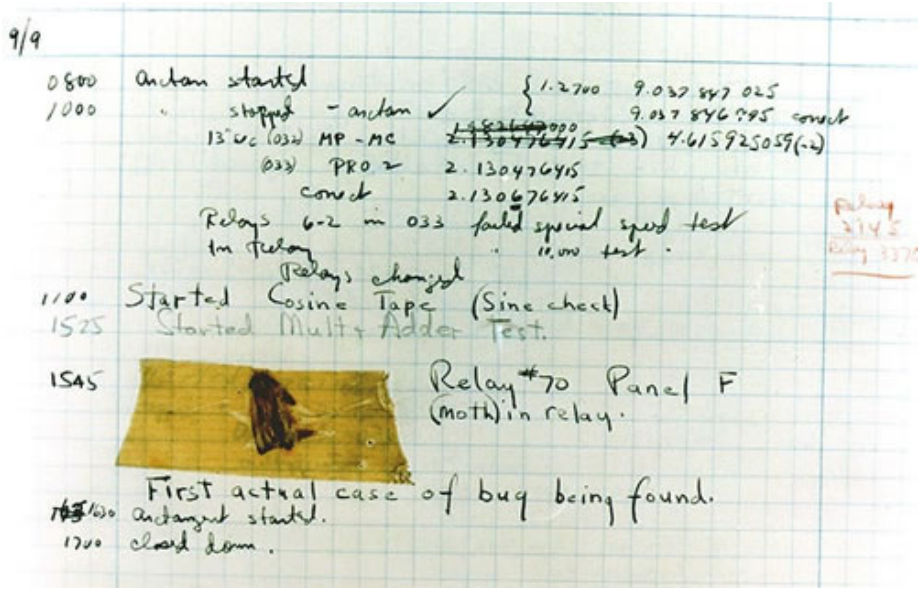
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
   setRandomNumberSeedFromTime();
   generateRandomNumberFromOTo9();
   takeUserInput();
   compareAndShowAnswer();
   return 0;
}
```

Functions

Purposes of functions:

- Increase module reusability
  Avoid re-inventing what you invented before!
- Increase module readability
  Can you understand your program after a month? A year?



- Reduce chances of creating a bug

- If you need to do the similar thing in multiple places in your code, you probably want to make it a function.

Functions

Let's be a little more realistic:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(void)
{
   int answer,guess;
   setRandomNumberSeedFromTime();
   answer=generateRandomNumberFromOTo9();
   guess=takeUserInput();
   compareAndShowAnswer(answer,guess);
   return 0;
}
```

Generated random number and user input needs to be stored somewhere.

1. Computer comes up with a random number.
2. You make a guess.
3. The computer will tell you if your guess is right or wrong.

Functions

Working code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void setRandomNumberSeedFromTime(void)
{
    int currentTime;
    currentTime=time(NULL);
    srand(currentTime);
}

int generateRandomNumberFrom0To9(void)
{
    int r;
    r=rand();
    r=r%10;
    return r;
}

int takeUserInput(void)
{
    int num;
    printf("Make a guess (0 to 9):");
    scanf("%d",&num);
    return num;
}

void compareAndShowAnswer(int answer,int guess)
{
    if(answer==guess)
    {
        printf("You've got the right answer!\n");
    }
    else
    {
        printf("Wrong.  The answer is %d\n",answer);
    }
}

int main(void)
{
    int answer,guess;

    setRandomNumberSeedFromTime();
    answer=generateRandomNumberFrom0To9();
    guess=takeUserInput();
    compareAndShowAnswer(answer,guess);
    return 0;
}
```

Functions

```
void setRandomNumberSeedFromTime(void)
{
    int currentTime;
    currentTime=time(NULL);
    srand(currentTime);
}
```

Takes no input from outside

Returns nothing to outside

It changes the seed number of the random-number generator, but the outside of this function does not need to know what seed number is set. Also, the new seed is taken from the timer. Therefore, this function does not need to take input from the outside.

By the way, it can be made shorter.

```
void setRandomNumberSeedFromTime(void)
{
    srand(time(NULL));
}
```

Functions

```
int generateRandomNumberFrom0To9(void)
{
    int r;
    r=rand();
    r=r%10;
    return r;
}
```

Returns an integer

Take no input from outside (can also leave blank)

Return value

This function does not take input from outside. What's necessary is taken from the random-number generator. The random number is then returned to whoever called it.

This can also be made shorter:

```
int generateRandomNumberFrom0To9(void)
{
    return rand()%10;
}
```

Functions

Returns nothing

```
void compareAndShowAnswer(int answer,int guess)
{
    if(answer==guess)
    {
        printf("You've got the right answer!\n");
    }
    else
        printf("Wrong.  The answer is %d\n",answer);
}
```

Take two integers as input

Curly brackets not needed if there is only one statement

Show messages depending on the correct answer and the guess. Since output is shown on the console window, no value needs to be returned to the outside.