When you haven't changed anything about your code but still hit run

Functions

CS2308
Gentry Atkinson

Mathematical Foundation

Algebraic Function

f(x) = 2x + 2

Input: x

Output: 2x+2

Example:

f(2) = 2*2 + 2

= 6

C++ Function

```
int f(int x){
   return 2*x + 2;
}
```

Input: int x

Output: 2*x+2

Example:

f(3) = 2*3 + 2

= 8

C++ Functions

- Define a relationship between inputs (parameters) and outputs (return statement).
- Every function returns one or zero values.
- A function can have any number of inputs.
- We must give a datatype for every input and output.

Example 1

```
int double_int (int x){
    return 2*x;
}

int main(int argc, char** argv){
    cout << "2 * 2 is " << double_int(2) << endl;
} //try to predict the output
```

Function Returns

- **Functions can have several return statements but will only return one value.**
- **Functions that do not return a value have the return type “void”.**

Example 2

```
int foo(int a){
    if (a < 2)
        return 0;
    else if (a < 10)
        return 5*a;
    else
        return a/10;
}

int main(int argc, char** argv){
    cout << "foo(a) is " << foo(2) << endl;
} //try to predict the output
```

Parameters vs. Arguments

- **Parameters: variables that are created as part of a function definition.**
- **Arguments: variables or literals that are "passed to" a function call.**
- **The const keyword can be used to keep a function from changing the value of its parameters.**

Example 3

```
void foo(string param){
    cout << "param was passed" << param << endl;
    return;
}

int main(int argc, char** argv){
    cout << "calling foo with the argument \"bar\"\n";
    foo("bar");
    return 0;
} //try to predict the output
```

Scope

- **Scope:** the portion of a program where a variable can be legally referenced.
- In general, variables are only “in scope” in the {block of code} that they were created in.
- Variables might still be in memory even when they are not in scope.
- 3 Levels of scope:
  - Global
  - Function
  - Block

Example 4

```
void foo(){
    int a = 3;
    cout << "a in foo is " << a << endl;
    return;
}

int main(int argc, char** argv){
    int a = 2;
    cout << "a in main is " << a << endl;
    foo();
    return 0;
} //try to predict the output
```

Function Prototypes

- **Adding many function definitions to a program can make the main function hard to find.**
- **To avoid this, we can use function prototypes to "warn" the compiler that a function will be used before it is defined.**
- **Prototype parameters do not have to have names.**

Example 5

```
void foo(int);

int main(int argc, char** argv){
    cout << "Calling foo with argument 2" << endl;
    foo(2);
    return 0;
} //try to predict the output

void foo(int a){
    cout << "foo passed " << a << endl;
    return;
}//try to predict the output
```

Questions or Comments?