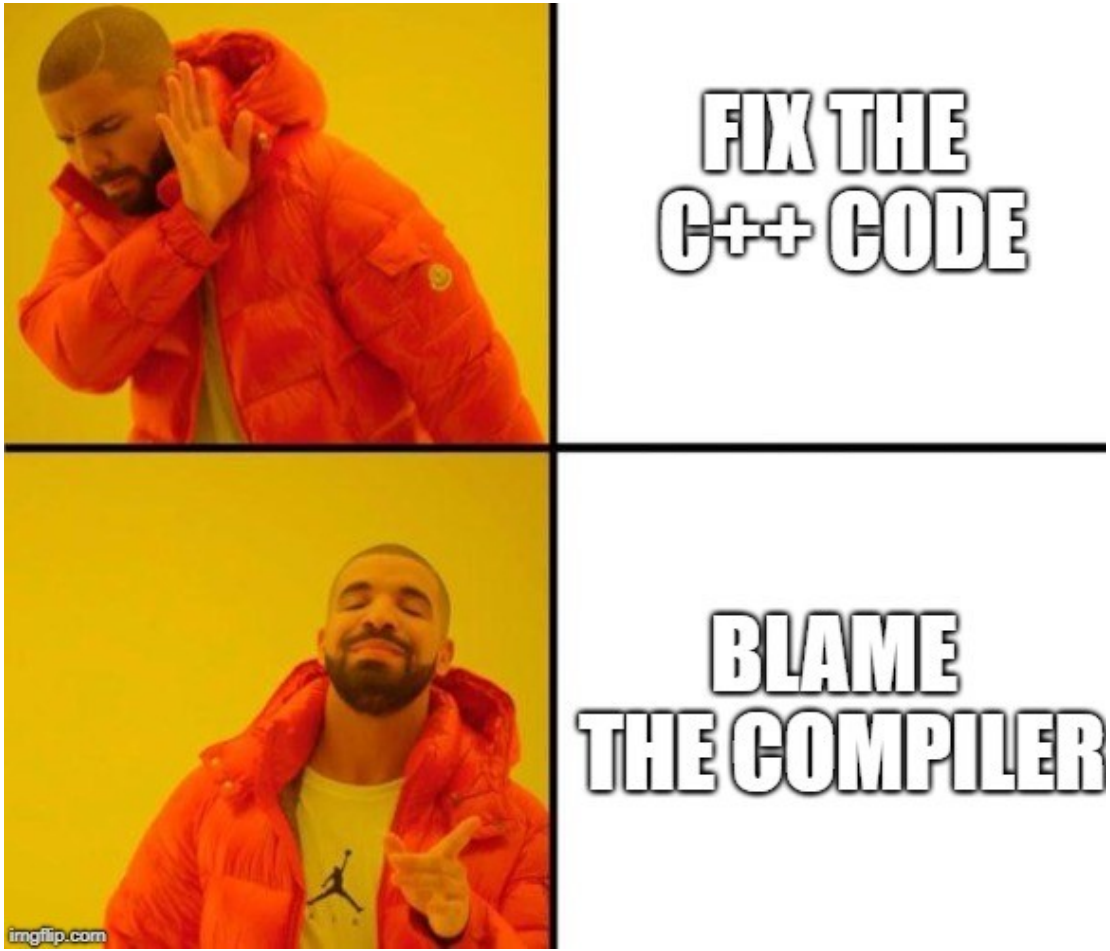
FIX THE
C++ CODE
BLAME
THE COMPILER
imgflip.com

CS1428
Foundation of Computer Science

Lecture 6: Functions

Let’s Think about Algebra

$f(x) = 2x + 2$

$f(4) = 2*4 + 2$
$f(4) = 10$

$y = 3$
$f(y) = 2*3 + 2$
$f(y) = 8$

$g(x) = 3x$
$f(g(5)) = f(3*5)$
$f(15) = 32$

Functions in C++

- Just like pure math, C++ let’s us define functions which we can use over and over again.
- Every function returns **one** value with a defined datatype.
- Every function has a list of variables that it expects to be passed, called the parameters.
- The parameters of a function are only in scope for that function.
- Program execution always begins in the **main** function.

A Very Simple Function

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

int main(){
    cout << double_a_number(4) << endl;
    return 0;
}
```

Simple Function Output

```
/home/gentry/Desktop/1428_testDir/junk

8

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Function Parameters

- The parameters are the best way to send information into a function.
- The parameters are declared in the parenthesis of the function definition.
- The parameters are local variables, so their name does not have to match the variable names in **main** but can.

Function Return

- Every function has a return type. Functions that do not return any value have the type “**void**”.
- The value that is returned by the function must match the return type.
- Even **void** functions should have a return statement (although the compiler might let you ignore it).

Return Type

```
char nextLetter(char a){
    if(a=='a') return ++a;
    else return 77;
}


int main(){
    cout << "Funciton return for 'a': "<< nextLetter('a') << endl;
    cout << "Function return for 'b': "<< nextLetter('b') << endl;
    return 0;
}
```

Function Output

```
/home/gentry/Desktop/1428_testDir/junk
Funciton return for 'a': b
Function return for 'b': M

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Variable Names Don’t Have to Match (but can)

```
int add_ints(int numberOne, int numberTwo){
    return numberOne + numberTwo;
}

int main(){
    int numberOne = 5;
    int b = 2;
    cout << "Sum of " << numberOne << " and " << b << " is "
        << add_ints(numberOne, b) << endl;
    return 0;
}
```

Function Output

```
/home/gentry/Desktop/1428_testDir/junk
Sum of 5 and 2 is 7

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Arguments vs. Parameters

- **Parameters** are declared in the function definition and are scoped locally to the called function.
- **Arguments** are passed to the function when it is called. They are locally scoped to the calling function. Arguments can be variables, constants, or literal.
- The value of arguments are copied into the parameter variables when the function is called.

Arguments vs. Parameters

Parameters

```
int add_ints(int numberOne, int numberTwo){
    return numberOne + numberTwo;
}

int main(){
    int numberOne = 5;
    int b = 2;
    cout << "Sum of " << numberOne << " and " << 4 << " is "
        << add_ints(numberOne, 4) << endl;
    return 0;
}
```

Arguments

Functions Don't "Remember" Variables

The variable sum is re-declared every time this function runs.

```
int add_ints(int numberOne, int numberTwo){
    int sum = numberOne + numberTwo;
    return sum;
}

int main(){
    count << "1+2=" << add_ints(1, 2) << endl;
    cout << "3+4=" << add_ints(3, 4) << endl;
    return 0;
}
```

Function Output

```
/home/gentry/Desktop/1428_testDir/junk
1+2=3
3+4=7

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Functions Remember ‘static’ Variables

```
int counting_function(){
    static int times_run = 0;
    times_run++;
    return times_run;
}

int main(){
    cout << "Run number: " << counting_function() << endl;
    cout << "Run number: " << counting_function() << endl;
    cout << "Run number: " << counting_function() << endl;
    cout << "Run number: " << counting_function() << endl;
    return 0;
}
```

Function Output

```
/home/gentry/Desktop/1428_testDir/junk
Run number: 1
Run number: 2
Run number: 3
Run number: 4

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

The static Keyword

- Keeps a function variable from being re-declared. Instead the declaration and initialization happen the first time the function is called in a program.
- The variable stays in memory for as long as the program is still running. Normally, functions variables are freed when the function returns.
- Global variables can also be static (it works differently but we don’t use global variables anyway).

The Danger of Global Variables

```
int loop_counter;

int loopy_function(int a){
    loop_counter = 0;
    while(loop_counter<6){loop_counter++; a++;}
    return a;
}

int main(){
    loop_counter = 0;
    while(loop_counter < 5){
        cout << "Loopy funciton: " << loopy_function(1)
            <<endl;
    }
    cout << "Loop counter: " << loop_counter<<endl;
}
```

Function Output

```
/home/gentry/Desktop/1428_testDir/junk
Loopy funciton: 7
Loop counter: 6

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Functions can be….

- Conditions
- The right hand side of assignments
- An operand in an operation
- A standalone statement

Function as a Condition

```
bool is_greater_than_5(int a){
    return a>5;
}

int main(){
    int loop_counter = 7;
    while(is_greater_than_5(loop_counter)){
        cout << "Loop counter: " << loop_counter<<endl;
        loop_counter--;
    }

}
```

Function with Assignment

```
int increment(int a){
    return a+1;
}

int main(){
    int loop_counter = 0;
    while(loop_counter<5){
        cout << "Loop counter: " << loop_counter<<endl;
        loop_counter = increment(loop_counter);
    }

}
```

Function in Operation

```
int new_value(){
    static int counter = 0;
    counter++;
    return counter;
}

int main(){
    int loop_counter = 1;
    while(loop_counter<5){
        cout << "Loop counter: " << loop_counter<<endl;
        loop_counter = 1 + new_value();
    }

}
```

Function as a Statement

```
void print_counter(int a){
    cout << "Loop Counter: " << a << endl;
    return;
}

int main(){
    int loop_counter = 1;
    while(loop_counter<5){
        print_counter(loop_counter);
        loop_counter++;
    }
    return 0;
}
```

You Can Define Many Functions!

```
void print_a(){
    cout << 'a';
    return;
}

void print_b(){
    cout << 'b';
    return;
}

void print_c(){
    cout << 'c';
    return;
}

void print_d(){
    cout << 'd';
    return;
}
```

The main function has been pushed out of sight.

Function Prototypes

- Adding so many functions that the **main** function is no longer visible in the first page of a program makes hard to read programs.
- Prototypes are short function signatures that warn the compiler that a function will be used, but don’t define the function.
- Using function prototypes lets us push the function definitions to below the **main** function.
- Prototypes are optional in this class, but are a good practice.

Function Prototypes

```
void print_B();
void print_k();
void print_o();


int main(){
    print_B();
    print_o();
    print_k();
    print_o();
    return 0;
}
```

```
void print_B(){
    cout << 'B';
    return;
}

void print_k(){
    cout << 'k';
    return;
}
void print_o(){
    cout << 'o';
    return;
}
```

Function Output

```
/home/gentry/Desktop/1428_testDir/junk
Boko
Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Prototypes with Parameters

```
void print_a_sum(int, int);

int main(){
    cout << "3+5=";
    print_a_sum(3,5);
    return 0;
}

void print_a_sum(int a, int b){
    cout << a+b << endl;
    return;
}
```

Function Output

```
/home/gentry/Desktop/1428_testDir/junk
3+5=8

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Function Overloading

- Several functions in the same program can have the <u>same name</u>.
- The “signature” of a function is the combination of its name and its parameters’ data types.
- Every function must have a unique signature.
- We can write several versions of a function to allow for flexibility for our users.

Overloaded Function

```
int add_numbers(int, int);
int add_numbers(int, int, int);

int main(){
    cout << "3+4=" << add_numbers(3, 4) << endl;
    cout << "3+4+5=" << add_numbers(3, 4, 5) << endl;
    return 0;
}

int add_numbers(int a, int b){
    return a+b;
}
int add_numbers(int a, int b, int c){
    return a+b+c;
}
```

Function Output

```
/home/gentry/Desktop/1428_testDir/junk
3+4=7
3+4+5=12

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```