UCLA Samueli
Computer Science

CS31 Week 4 Discussion

Fall 2021, Section 1C
Mingyu Derek Ma mdma@ucla.edu
Thanks Muhao Chen, Rosa Garza for their shared content

https://derek.ma/cs31 for slides and other discussion materials

Reminder

- Midterm, Oct 26 Tue 6:30pm
- Project 3, Oct 27 Wed 11pm
- Video to solve last week’s worksheet

Project 2 Grading

- Report
  - Test cases should cover different range of valid input and invalid input
- Warm up/Homework
  - Make sure to meet the question requirement: like using a simple English sentence to describe the program, no need to provide too much details/examples/internal variables
- Program
  - Comment to explain your program logic
  - Especially for the if-else conditions and formula
  - Explain the magic numbers by variable names or comments

Mid-term

- UPE’s midterm review session, Oct 25 7pm
- Past exam problems and solutions by Kung-Hua Chang
- Additional links and past exams on our website: derek.ma/cs31

Project 3 Suggestions

- Read spec and FAQ thoroughly
- Go through writeups in Oct 17 announcement!
- Write down your program design first
  - What do you want to achieve for each part of your code
  - What variables will be updated/changed in a particular part
  - Use comment to layout your design and have some very simple function placeholders there
- Develop incrementally
  - Implement a simple case, then expand to more conditions
- Save your immediate versions
  - You can compare and revert if something doesn’t work
  - You have something to turn in if you run out of time!

Function

- Input some parameters to the function, run a batch of statements, return a result
- Declare the type of the return value
- Declare input parameters (local variables only available in this function)
- Call a function by its name and pass in values

```
int max(int a, int b) {
        if (a > b) return a;
        return b;
}

int main(){
        int a;
        a = max(4, 6)
}
```

void

- A function with a void return type means no return value
- Cannot assign return value from a void function to a variable since there is no return value

```
void example1Func1(){
    cout << "We are now inside example1Func1" << endl;
}

void example1(){
    int a;
    // The following statement is wrong
    // Error: Assigning to 'int' from incompatible type 'void'
    a = example1Func1();
}
```

Assigning to 'int' from incompatible type 'void'

Everything inside a function is local

- Variables defined in the function cannot be used outside the function

```
void printFactorial(int n) {
    int prod = 1;
    for (int i = 2; i <= n; i++)
        prod *= i;
    cout << "The factorial of " << n << " is " << prod << endl;
}

void example2(){
    printFactorial(4);
    cout << prod;
    cout << n;
    cout << i;
}
```

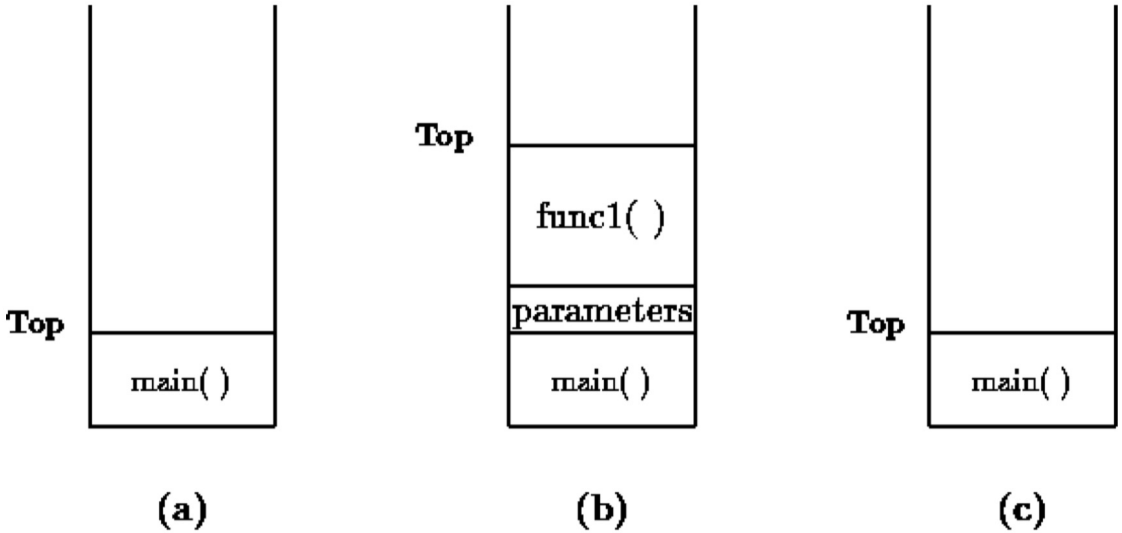
2 ⚠ ⊙ Use of undeclared identifier 'prod'; did you mean 'stod'?

⊗ Use of undeclared identifier 'n'

⊗ Use of undeclared identifier 'i'

Everything inside a function is local

- Variables defined in the function cannot be used outside the function



Do Not: overwrite arguments

- Do not define local variables using the same name as arguments -> compile error
- Warning: the parameters can be overwritten

```
int example3Func1(int a, int b){
    int a; // Error: Redefinition of 'a'   Redefinition of 'a'
    double b; // Error: Redefinition of 'b' with a
        different type
    cout << a << endl;
    cout << b << endl;
    return a + b;
}

void example3(){
    int a;
    a = example3Func1(4, 6); // compile error
    cout << a << endl;
}
```

Do Not: overwrite arguments

- Do not define local variables using the same name as arguments -> compile error
- Warning: the parameters can be overwritten

```
int example3Func2(int a, int b){
    a = 7;
    b = 10;
    cout << a << endl; // 7
    cout << b << endl; // 10
    return a + b;
}

void example3(){
    int a;
    a = example3Func2(4, 6); // this a is 17
    cout << a << endl;
}
```

Take care of no return of a function

- If certain condition leads to no return, then it's a compile error or undefined behavior
- Example: if `a` is `0`, there is no return

```
bool non_negative(int a){
    if(a > 0)
        return true;
    else if(a < 0)
        return false;
}           ⊗ Non-void function does not return a value in all control paths

void example4(){
    non_negative(4);
}
```

Take care of inconsistent return type

- `c` should be 0.75, a `double`
- But the returned value is 0 since the return data type of this function is defined as `int`
- Will result in logic errors

```
int example5Func1(int a, int b){
    double c = 1.0 * a / b;
    cout << c << endl;
    return c;
}

int main(){
    int a = example5Func1(3, 4);
    cout << a << endl;
}
```

```
0.75
0
```

Functions of the same name: Function overloading

- Different functions can use same function names if they have different type of arguments OR number of arguments
- Functions with return type difference only cannot be overloaded

Functions of the same name: Function overloading

```
int print(int a){
    cout << "Inside print 1" << endl;
    return 0;
}

int print(double a){
    cout << "Inside print 2" << endl;
    return 0;
}

//double print(int a){
//    cout << "Inside print 3" << endl;
//    return 0;
//}

int print(int a, int b){
    cout << "Inside print 4" << endl;
    return 0;
}

int main(){
    cout << print(2) << endl;
    cout << print(3.7) << endl;
    cout << print(3,4) << endl;
}
```

```
Inside print 1
0
Inside print 2
0
Inside print 4
0
```

Functions of the same name: Function overloading

```
int print(int a){
    cout << "Inside print 1" << endl;
    return 0;
}

int print(double a){
    cout << "Inside print 2" << endl;
    return 0;
}

double print(int a){   Functions that differ only in their return type cannot be overloaded
    cout << "Inside print 3" << endl;
    return 0;
}

int print(int a, int b){
    cout << "Inside print 4" << endl;
    return 0;
}

int main(){
    cout << print(2) << endl;
    cout << print(3.7) << endl;
    cout << print(3,4) << endl;
}
```

Where to place your function

- Place function before main

```
int example7Func1(int a){
    cout << "Inside example7Func1" << endl;
    return a;
}

int main(){
    int a = example7Func1(3);
    cout << a << endl;
}
```

```
int main(){
    int a = example7Func1(3);   Use of undeclared identifier 'example7Fun...
    cout << a << endl;
}

int example7Func1(int a){
    cout << "Inside example7Func1" << endl;
    return a;
}
```

Where to place your function

- Place function before main
- Otherwise need to declare the function first

```
int example7Func1(int);

int main(){
    int a = example7Func1(3);
    cout << a << endl;
}

int example7Func1(int a){
    cout << "Inside example7Func1" << endl;
    return a;
}
```

Pass by reference vs pass by value

- Pass by value to a function won't change the values of the variables used to pass the value
  - Representation: `a`
  - Read-only
  - Copy the values of the parameters to local variables
- Pass by reference: we want the function to operate on the parameters directly
  - Representation: `&a`
  - Read-or-write
  - Consider it as another name for an already existing object
  - No copying, directly pass the addresses

Pass by reference vs pass by value

```
void swapValue(int a, int b){
    cout << "==== in function swapValue" << endl;
    int tmp = a;
    a = b;
    b = tmp;
    cout << "tmp is " << tmp << endl;
    cout << "a is " << a << endl;
    cout << "b is " << b << endl;
}

void swapReference(int &a, int &b){
    cout << "==== in function swapReference" << endl;
    int tmp = a;
    a = b;
    b = tmp;
    cout << "tmp is " << tmp << endl;
    cout << "a is " << a << endl;
    cout << "b is " << b << endl;
}
```

```
int main(){
    int x = 4, y = 6;
    swapValue(x, y);
    cout << "x is " << x << endl;
    cout << "y is " << y << endl;
    swapReference(x, y);
    cout << "x is " << x << endl;
    cout << "y is " << y << endl;
}
```

```
==== in function swapValue
tmp is 4
a is 6
b is 4
x is 4
y is 6
==== in function swapReference
tmp is 4
a is 6
b is 4
x is 6
y is 4
```

Return; Break; Continue;

- Return: terminates a **function**
- Break: terminates a **loop**
- Continue: terminates **a cycle of a loop**

Return; Break; Continue;

```
void example9Func1(string s){
    for (int i=0; i<s.size(); ++i){
        if(islower(s.at(i)))
            return;
        cout << s.at(i);
    }
    cout << "Nismo";
}

void example9Func2(string s){
    for (int i=0; i<s.size(); ++i){
        if(islower(s.at(i)))
            break;
        cout << s.at(i);
    }
    cout << "Nismo";
}
```

```
void example9Func3(string s){
    for (int i=0; i<s.size(); ++i){
        if(islower(s.at(i)))
            continue;
        cout << s.at(i);
    }
    cout << "Nismo";
}

int main(){
    string inputString = "Nissan GTR";
    example9Func1(inputString);
    // Output is N
    example9Func2(inputString);
    // Output is NNismo
    example9Func3(inputString);
    // Output is N GTRNismo
}
```

Thank You