CISC 1600/1610 Computer Science I

Functions, pass-by-reference

Professor Daniel Leeds
dleeds@fordham.edu
JMH 328A

Review

- `printOut` – void and overloading
- `rand` – random numbers -> random letters

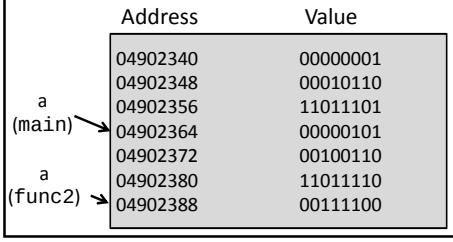
What does this code do?

```
int newFunc(int a);

int main() {
  int a=5, b, c=5;
  b = newFunc(a);
  cout << a << " " << b << " "
       << c << endl;
  return 0;
}

int newFunc(int a) {
  a*=12;
  return a+5;
}
```


Memory usage by functions

"Call-by-value":

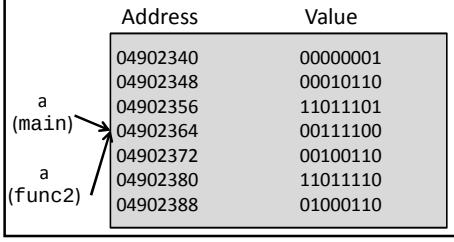
- provide function with the value held in a variable input
- assign value to new internal variable



Memory usage by functions

"Call-by-reference":

- provide function with the **address** of a variable input
- assign value into old address



Call-by-Reference Syntax

- Use & to indicate a variable is called by reference
- Use & both in declaration and definition
- Don't use & in call

```
void get_letters(char& letter1, char& letter2);

...

void get_letters(char& letter1, char& letter2)
{
   cout << "Enter two letters: ";
   cin >> letter1 >> letter2;
}
```

Call-by-Reference Syntax

```
void get_letters(char& letter1, char& letter2);

int main() {
  char a,b;
  get_letters(a,b);
  cout << a << " " << b << endl;
  return 0;
}

void get_letters(char& letter1, char& letter2)
{
   cout << "Enter two letters: ";
   cin >> letter1 >> letter2;
}
```

7

Call-by-reference vs. Call-by-value

- Call-by-value preserves the value of the original input argument
- Call-by-reference can change the value of the original input argument
  - Effectively allows return of multiple values from function

8

```
int mysteryFunc(int& num1);

int main() {
  int a=5;
  cout << mysteryFunc(a) << endl;
  cout << a << endl;
  return 0;
}

int mysteryFunc(int &num1) {
  num1 += 3;
  return num1/4;
}
```

What does this do?

9

```
int mysteryFunc2(int inNum);

int main() {
  int a=3;
  cout << mysteryFunc2(a);
  cout << a;
  return 0;
}

int mysteryFunc2(int inNum) {
  inNum = inNum*inNum;
  return inNum;
}
```

What does this do?

10

Call-by-reference: Input arguments

- Arguments must be variables
  If declare: `void myFunc(float& inputNum);`
  - `myFunc(inVariable);` - GOOD syntax
  - `myFunc(25.4);` - BAD syntax

11

Mixing parameters

- Can define a function that takes both values and references

```
void flipAndMult(int& num1, int& num2, int mult);
// flips num1 and num2 and multiplies each
// by mult
```

12

More usage of &

```
int x = 5;
int& y=x; // y and x point to same address
y=10;
cout << x << endl; // output x value
cout << &x << endl; // output x address
```

13

Broader scope: global variables

- Global variables visible to all functions
- Declared outside of all functions
- Must be declared prior to first use

```
#include<iostream>
using namespace std;
const float PI=3.14;
    // visible to main and to areaCircle

// compute area of circle
float areaCircle(float radius);

int main() { ...}
float areaCircle(float radius) {...}
```

14

More on global variables

- Useful to define global constants
- Very risky to define non-constant global variables
  - try to keep track of what functions change the variable

15