C Function Invocation

The C Programming Language, Chapter 4
int main(int argc, char **argv) {
    int a=3;
    int b=4;
    int c = myfn(a,b+2);
    return 0;
}

int myfn(int x, int y) { return x*x + y*y; }
Terminology Definitions

• Caller: Higher level function that invokes a lower level function
• Callee: Lower level function that has been invoked by a higher level function
  • Note: A single function can be both a callee and a caller
• Invocation: The indication in the caller that it want’s to use the callee
• Instance: One specific execution of the callee
• Return Value: The value specified on the “return” statement of the callee
• Return Address: The instruction in the caller that should be executed when the instance of the callee is complete
Instruction processing...

[OS(caller) invokes Main (callee)]
[Main (callee) startup]

```plaintext
a=3;
b=4;
```

[Main (caller) invokes myfn(callee)]

```plaintext
x=a, y=b+2;
```

[myfn (callee) startup]

```plaintext
x*x + y*y
```

[myfn (callee) return to main]

[main (caller) return from myfn]

```plaintext
c = [return value from myfn]
```

[main (callee) return to OS]

```c
int main(int argc, char **argv) {
    int a=3;
    int b=4;
    int c = myfn(a,b+2);
    return 0;
}
```

```c
int myfn(int x, int y) { return x*x + y*y; }
```
Function Invocation Details

[OS(caller) invokes Main (callee)]
[Main (callee) startup]

\[a=3; \quad b=4;\]

[Main (caller) invokes myfn(callee)]
\[x=a, \quad y=b+2;\]
[myfn (callee) startup]
\[x^2 + y^2\]

[myfn (callee) return to main]
[main (caller) return from myfn]
\[c = \text{[return value from myfn]}\]
[main (callee) return to OS]
int main(int argc, char **argv) {
    int a = 3;
    int b = 4;
    int c = myfn(a, b + 2);
    return 0;
}

int myfn(int x, int y) { return x*x + y*y; }
Callee startup...

```c
int main(int argc, char **argv) {

    int a=3;
    int b=4;
    int c = myfn(a,b);
    return 0;
}

int myfn(int x, int y) { return x*x + y*y; }
```
Function Return Details

[OS(caller) invokes Main (callee)]
[Main (callee) startup]

a=3;
b=4;

[Main (caller) invokes myfn(callee)]
x=a, y=b+2;

[myfn (callee) startup]

x*x + y*y

[myfn (callee) return to main]
[main (caller) return from myfn]

c = [return value from myfn]

[main (callee) return to OS]
Callee return...

```c
int main(int argc, char **argv) {
    int a=3;
    int b=4;
    int c = myfn(a,b);
    return 0;
}

int myfn(int x, int y) { return x*x + y*y; }
```
int main(int argc, char **argv) {
    int a = 3;
    int b = 4;
    int c = myfn(a, b);
    return 0;
}

int myfn(int x, int y) { return x*x + y*y; }
“Call by Value”

• Convention that says callee works on a copy of the arguments

• Protects callers variables from unexpected “side-effects”
  int x=1,y=2,z=3;
  int myfunc(int a; int b) { a=2*a; return a*b; }
  x=myfunc(y,z);

• Prevents functions from modifying arguments
  void increment(int a) { a=a+1; }
  increment(x);
Update Argument work-arounds

• Use the return value

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

• Pass a pointer to the value

```c
void increment(int *a) { (*a) = (*a) + 1; }
increment(&x);
```
Function Variable Scope

- Variables are visible only inside the block in which they are declared
  - Block is delimited by curly braces... {   }
  - Variables typically declared inside a function definition
  - Not visible to called functions! (Different from other languages)

- That way, “x” in this function is always different from “x” in the calling function!

- Function variables disappear when the block ends
Example Scope Hole

{ int i; int j=7;
  for (i=0; i<3; i++) {
    int j=i+1;
    printf("j=%d \n",j);
  }
  printf("j=%d \n");
}
Function Binding in C

- Binding: How much do I need to know in order to code a function
  - Parameter Definitions
  - Return Value Type
  - Functionality
  - Environment (global variables, lower level function availability, etc.)
  - Caller’s environment (variable names, etc.)
  - State (status of IO devices, value of caller’s variables, etc.)

- High Binding => Low Portability / Re-Use
- Function Variables in C => Lower Binding
Variable Life

• Global Variables (declared outside function):
  • allocated/initialized before “main” is invoked
  • freed after “main” returns

• Local Variables:
  • allocated/initialized when function is invoked
  • freed after function returns
The “Disappearing” return value problem

```c
char * myfn(int x) {
    char val[256];
    int j;
    for(j=0; j<x && j<256; j++) val[j] = 'x';
    val[j] = 0;
    return val;
}

int main() {
    char *p = myfn(121);
    printf("121 xs are: %s\n", p);
    // Problem here!
    return 0;
}
```
return value fix

char val[256]; // “Global” variable
char * myfn(int x) {
    int j;
    for(j=0;j<x && j<256; j++) val[j] = 'x';
    val[j] = 0; return val;
}

int main() {
    char *p = myfn(121); // p alias for “val”
    printf("121 xs are: %s\n", p); // Works, but dangerous!
    return 0;
}
return value fix 2

char * myfn(int x) {
    int j;
    char *val=(char *)malloc(x+1); // get memory from OS
    for(j=0;j<x; j++) val[j]='x';
    val[j]=0; return val;
}

int main() {
    char *p=myfn(121);
    printf("121 xs are: %s \n",p);
    free(p); // return memory to OS
    return 0;
}