Scope

2024 Winter APS 105: Computer Fundamentals
Jon Eyolfson

Lecture 11
1.1.2

When We Use a Function, We Say “Function Call”

Given the previous code snippet:

```
int main(void) {
    /* Get the input. */
    printTriangle(n);
    return 0;
}
```

We would say `main` calls `printTriangle`,
C copies the arguments and jumps there

When We Use a Function, We Say “Function Call”

Given the previous code snippet:

```
int main(void) {
    /* Get the input. */
    printTriangle(n);
    return 0;
}
```

We would say `main` calls `printTriangle`,
C copies the arguments and jumps there

In this case `main` is the caller, and `printTriangle` is the callee

When We Use a Function, We Say “Function Call”

Given the previous code snippet:

```
int main(void) {
    /* Get the input. */
    printTriangle(n);
    return 0;
}
```

We would say `main` calls `printTriangle`,

C copies the arguments and jumps there

In this case `main` is the caller, and `printTriangle` is the callee

A `return` stops the callee, and the caller resumes

The caller gets a copy of the return value

The Scope is Part of the Program You Can Use a Variable

You can use a variable declaration within a { until the matching }

C declares function arguments, and `for` loop initializers in the next {

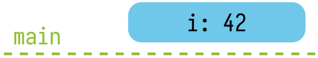
Variables Exist Within a Function

```
int main(void) {
    int i = 42;
    printf("i: %d\n", i);
    return 0;
}
```

Variables Exist Within a Function

```
int main(void) {
    int i = 42;
    printf("i: %d\n", i);
    return 0;
}
```

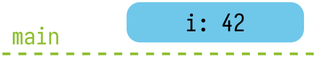
i is valid here



Variables Exist Within a Function

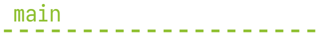
```c
int main(void) {
    int i = 42;
    printf("i: %d\n", i);
    return 0;
}
```

i is valid here



A For Loop Creates an Inner Scope

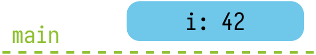
```
int main(void) {
→   int i = 42;
    for (int j = 0; j < 1; ++j) {
        printf("j: %d\n", j);
    }
    return 0;
}
```



A For Loop Creates an Inner Scope

```
int main(void) {
    int i = 42;
    for (int j = 0; j < 1; ++j) {
        printf("j: %d\n", j);
    }
    return 0;
}
```

i is valid here



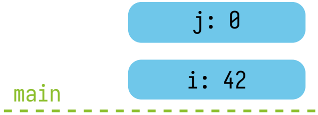
main

i: 42

A For Loop Creates an Inner Scope

```c
int main(void) {
    int i = 42;
    for (int j = 0; j < 1; ++j) {
        printf("j: %d\n", j);
    }
    return 0;
}
```

j is valid here



j: 0

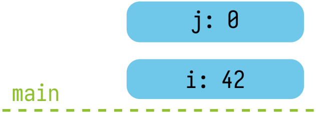
i: 42

main

A For Loop Creates an Inner Scope

```
int main(void) {
    int i = 42;
    for (int j = 0; j < 1; ++j) {
        printf("j: %d\n", j);
    }
    return 0;
}
```

j is valid here



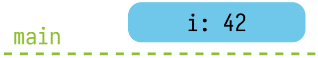
j: 0

i: 42

main

A For Loop Creates an Inner Scope

```
int main(void) {
    int i = 42;
    for (int j = 0; j < 1; ++j) {
        printf("j: %d\n", j);
    }
    return 0;
}
```



You Can “Shadow” A Variable with the Same Name

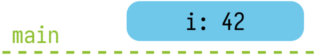
```
int main(void) {
    int i = 42;
    for (int i = 0; i < 1; ++i) {
        printf("i: %d\n", i);
    }
    return 0;
}
```

main

You Can “Shadow” A Variable with the Same Name

```
int main(void) {
    int i = 42;
    for (int i = 0; i < 1; ++i) {
        printf("i: %d\n", i);
    }
    return 0;
}
```

i is valid here



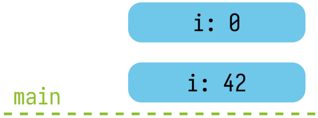
main

i: 42

You Can “Shadow” A Variable with the Same Name

```
int main(void) {
    int i = 42;
    for (int i = 0; i < 1; ++i) {
        printf("i: %d\n", i);
    }
    return 0;
}
```

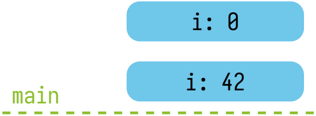
i is valid here (C always uses the most recent)



You Can “Shadow” A Variable with the Same Name

```
int main(void) {
    int i = 42;
    for (int i = 0; i < 1; ++i) {
        printf("i: %d\n", i);
    }
    return 0;
}
```

i is valid here (C always uses the most recent)



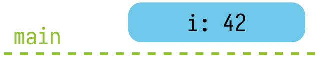
i: 0

i: 42

main

You Can “Shadow” A Variable with the Same Name

```c
int main(void) {
    int i = 42;
    for (int i = 0; i < 1; ++i) {
        printf("i: %d\n", i);
    }
→   return 0;
}
```



A Set of Curly Brackets Is Also a New Inner Scope

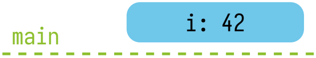
```
int main(void) {
→   int i = 42;
    {
        int j = 0;

    }
    return 0;
}
```

main

A Set of Curly Brackets Is Also a New Inner Scope

```
int main(void) {
    int i = 42;
    {
        int j = 0;

    }
    return 0;
}
```

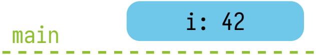
i valid



A Set of Curly Brackets Is Also a New Inner Scope

```
int main(void) {
    int i = 42;
    {
        int j = 0;

    }
    return 0;
}
```

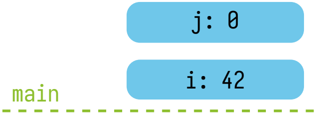
i valid



A Set of Curly Brackets Is Also a New Inner Scope

```
int main(void) {
    int i = 42;
    {
        int j = 0;

    }
    return 0;
}
```

j valid



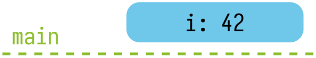
j: 0

i: 42

main

A Set of Curly Brackets Is Also a New Inner Scope

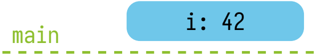
```c
int main(void) {
    int i = 42;
    {
        int j = 0;

→   }
    return 0;
}
```


main

i: 42

A Set of Curly Brackets Is Also a New Inner Scope

```c
int main(void) {
    int i = 42;
    {
        int j = 0;

    }
    return 0;
}
```



main

i: 42

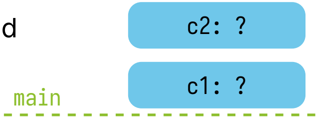
Functions Create Their Own Scope

```
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
→   char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

Functions Create Their Own Scope

```c
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

c1 and c2 valid



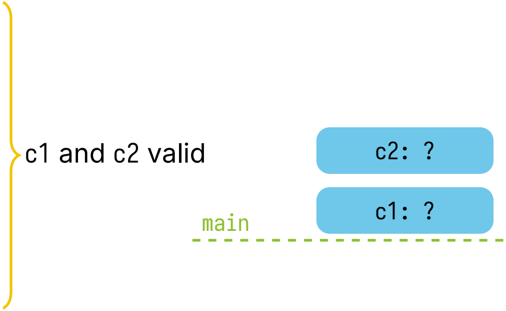
c2: ?

c1: ?

main

Functions Create Their Own Scope

```
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
→   scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```



c1 and c2 valid

c2: ?

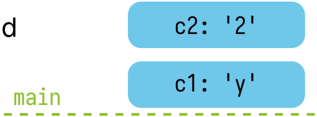
c1: ?

main

Functions Create Their Own Scope

```c
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

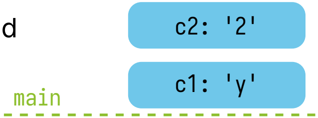
c1 and c2 valid



Functions Create Their Own Scope

```c
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

c1 and c2 valid



c2: '2'

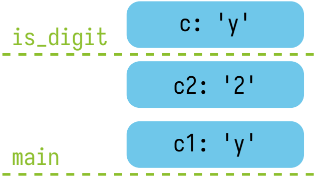
c1: 'y'

main

Functions Create Their Own Scope

```
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

only c valid



is_digit

c: 'y'

c2: '2'

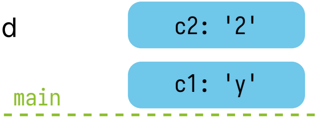
c1: 'y'

main

Functions Create Their Own Scope

```
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

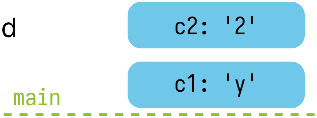
c1 and c2 valid



Functions Create Their Own Scope

```
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

c1 and c2 valid



c2: '2'

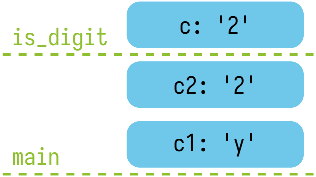
c1: 'y'

main

Functions Create Their Own Scope

```
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

only c valid



is_digit

c: '2'

c2: '2'

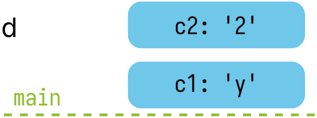
c1: 'y'

main

Functions Create Their Own Scope

```
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

c1 and c2 valid



c2: '2'

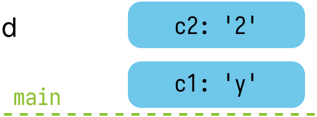
c1: 'y'

main

Functions Create Their Own Scope

```
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

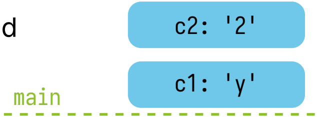
c1 and c2 valid



Functions Create Their Own Scope

```c
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

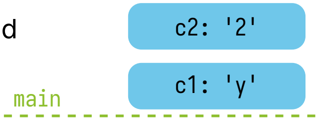
c1 and c2 valid



Functions Create Their Own Scope

```
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
→   printf("\n");
    return 0;
}
```

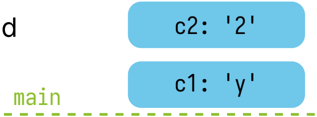
c1 and c2 valid



Functions Create Their Own Scope

```c
bool is_digit(char c) {
    return c >= '0' && c <= '9';
}

int main(void) {
    char c1, c2;
    printf("Input 2 characters: ");
    scanf(" %c %c", &c1, &c2);
    printf("Digits:");
    if (is_digit(c1)) {
        printf(" %c", c1);
    }
    if (is_digit(c2)) {
        printf(" %c", c2);
    }
    printf("\n");
    return 0;
}
```

c1 and c2 valid



Beware: `scanf` Behaves Strangely with Characters

If we write: `scanf("%c %c", &c1, &c2);`
and our input is: `" A B"`
then `c1 = ' '`; and `c2 = 'A'`;

Beware: `scanf` Behaves Strangely with Characters

If we write: `scanf("%c %c", &c1, &c2);`
and our input is: `" A B"`
then `c1 = ' '`; and `c2 = 'A'`;

If we write: `scanf("%c%c", &c1, &c2);`
and our input is: `"  A B"`
then `c1 = ' '`; and `c2 = ' '`;

Beware: `scanf` Behaves Strangely with Characters

If we write: `scanf("%c %c", &c1, &c2);`
 and our input is: `" A B"`
 then `c1 = ' '`; and `c2 = 'A'`;

If we write: `scanf("%c%c", &c1, &c2);`
 and our input is: `"  A B"`
 then `c1 = ' '`; and `c2 = ' '`;

Unless you really want to capture a space character,
 always put a space before `%c` and,
 never put a space after a format specifier

Beware: `scanf` Behaves Strangely with Characters

If we write: `scanf("%c %c", &c1, &c2);`
and our input is: `" A B"`
then `c1 = ' '`; and `c2 = 'A'`;

If we write: `scanf("%c%c", &c1, &c2);`
and our input is: `"  A B"`
then `c1 = ' '`; and `c2 = ' '`;

Unless you really want to capture a space character,
always put a space before `%c` and,
never put a space after a format specifier

From the real documentation:
“It is very difficult to use these functions correctly”...

Just In Case: `,` in an Expression is an Operator

The `,` operator will evaluate the expression on the left-hand side,
and throw the result away, then it evaluates the right-hand side,
and the result of the expression is the result of the right-hand side

You may see things like: `for (int x = 0, y = 10; x < y; ++x, --y)`

What Do We See Printed?

```c
#include <stdio.h>

void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
    printf("swap a: %d, b: %d\n", a, b);
}

int main(void) {
    int a = 1;
    int b = 2;
    printf("main (before swap) a: %d, b: %d\n", a, b);
    swap(a, b);
    printf("main (after swap)  a: %d, b: %d\n", a, b);
    return 0;
}
```

The Output of the Previous Program

```
main (before swap) a: 1, b: 2
swap a: 2, b: 1
main (after swap)  a: 1, b: 2
```

We Can Create Global Variables

A global variable is always in scope (valid) below its declaration

They’re created before `main` runs

We Can Create Global Variables

A global variable is always in scope (valid) below its declaration

They're created before `main` runs

They are not advised for this course, and you should avoid them

If you absolutely have to make globals, declare them as `static`
(so they can't accidentally be used in other C files)

Any Function Can Change Global Variables

```c
#include <stdio.h>

static int a = 1;
static int b = 2;

void swap(void) {
    int temp = a;
    a = b;
    b = temp;
    printf("swap a: %d, b: %d\n", a, b);
}

int main(void) {
    printf("main (before swap) a: %d, b: %d\n", a, b);
    swap();
    printf("main (after swap)  a: %d, b: %d\n", a, b);
    return 0;
}
```

We’ve Covered Functions, Now Practice!

We’ve now completed chapter 5 of the Learning C online book
  Try: `https://learningc.org/chapters/chapter05-functions/exercises`

There’s also past (midterm) exams:
`https://q.utoronto.ca/courses/330896/pages/past-exams`

Please feel free to discuss and ask questions in Discord!

The Real Swap Function (Next Lecture)

```
#include <stdio.h>
#include <stdlib.h>

void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main(void) {
    int a = 1;
    int b = 2;
    printf("main (before swap) a: %d, b: %d\n", a, b);
    swap(&a, &b);
    printf("main (after swap)  a: %d, b: %d\n", a, b);
    return EXIT_SUCCESS;
}
```

Bonus Practice: Factorial

Create a function called `factorial` that takes an integer argument, `n`, and returns an integer that's the result of computing n!

Recall: n! is
$n \times (n-1) \times (n-2) \times \cdots \times 1$

The factorial of a negative number is undefined

Example Previous Solution

```
int factorial(int n) {
    if (n < 0) {
        return 0;
    }
    int product = 1;
    for (int x = 1; x <= n; ++x) {
        product *= x;
    }
    return product;
}
```