

**CSc 352**

C Programming  
Arguments, Functions, Boolean

Benjamin Dicken

# Command line args

- Recall: Many programs can have ***command-line arguments***
- Common way to enable/disable settings, send information, give file names to a process that you want to start
- Three types

## **Flag**

A boolean option that begins with a dash (enable/disable feature)

## **Named argument**

A flag, with a value following

## **Positional argument**

An argument with no flag

# How to “get” these in a C program?

```
int main(●) {  
    . . .  
}
```

Stores the number of  
elements given on the  
command line

```
int main(int argc, char * argv[]) {  
    . . .  
}
```

A 2D char array, containing  
each component of the  
command

Standard practice to  
use these names for  
the arguments

# What will this print?

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

```
int main(int argc, char * argv[]) {  
    for (int i = 0; i < argc; i++) {  
        printf("element %d: %s\n", i, argv[i]);  
    }  
    return 0;  
}
```

```
$ ./a.out -v hi there
```

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
int main(int argc, char * argv[]) {
    bool help_enabled = false, case_enabled = false;
    for (int i = 0; i < argc; i++) {
        if (strcmp(argv[i], "-h") == 0) {
            help_enabled = true;
        } else if (strcmp(argv[i], "-c") == 0) {
            case_enabled = true;
        }
    }
    if (help_enabled) {
        printf("A help message\n");
        return 0;
    }
    if (case_enabled) {
        // run some code
    }
    // The rest of your main function....
    return 0;
}
```

## How to implement a flag argument in C

```
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
int main(int argc, char * argv[]) {
    bool help_enabled = false, case_enabled = false;
    for (int i = 0; i < argc; i++) {
        if (strcmp(argv[i], "-h") == 0) {
            help_enabled = true;
        } else if (strcmp(argv[i], "-c") == 0) {
            case_enabled = true;
        }
    }
    if (help_enabled) {
        printf("A help message\n");
        return 0;
    }
    if (case_enabled) {
        // run some code
    }
    // The rest of your main function....
    return 0;
}
```

What's the deal with these files that end in .h?

C11 does support a bool type, \*if\* you use stdbool.h

What is strcmp? How do I know which functions I can and should use?

# Functions

- The basic premise of function declaration, definition, calling, etc is similar to Java and Python
- Will discuss today - some nuances to understand here
- There are no *methods* in C, but there are a bunch of functions a part of the **C Standard Library** that we can leverage
- Let's look at defining our own functions, then investigate the standard library

```
#include <stdio.h>

int main(int argc, char * argv[]) {
    int z = greater(250, 300);
    printf("%d\n", z);
    custom_printf("output");
    return 0;
}

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

void custom_printf(char text[]) {
    printf("LOOK! %s\n", text);
}
```

A program with  
multiple  
functions



```
#include <stdio.h>

int main(int argc, char * argv[]) {
    int z = greater(250, 300);
    printf("%d\n", z);
    custom_printf("output");
    return 0;
}

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

void custom_printf(char text[]) {
    printf("LOOK! %s\n", text);
}
```

What does  
this do?

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


```
int greater(int a, int b);  
void custom_printf(char text[]);
```

```
int main(int argc, char * argv[]) {  
    int z = greater(250, 300);  
    printf("%d\n", z);  
    custom_printf("output");  
    return 0;  
}
```

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

```
void custom_printf(char text[]) {  
    printf("LOOK! %s\n", text);  
}
```

Can solve with  
a function  
prototype




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

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

```
void custom_printf(char text[]) {  
    printf("LOOK! %s\n", text);  
}
```

```
int main(int argc, char * argv[]) {  
    int z = greater(250, 300);  
    printf("%d\n", z);  
    custom_printf("output");  
    return 0;  
}
```

Or change  
the order



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

```
char[] get_a_string() {  
    char content[50];  
    printf("Please enter a string:\n");  
    scanf("%49s", content);  
    return content;  
}
```

```
int main(int argc, char * argv[]) {  
    char[] input;  
    input = get_a_string();  
    printf("characters: %s\n", input);  
    return 0;  
}
```

# What will this print out?

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

```
char[] get_a_string() {  
    char content[50];  
    printf("Please enter a string:\n");  
    scanf("%49s", content);  
    return content;  
}
```

```
int main(int argc, char * argv[]) {  
    char[] input;  
    input = get_a_string();  
    printf("characters: %s\n", input);  
    return 0;  
}
```

Cannot just return a  
char[] from a function

You *can* do something  
similar if you use a  
pointer to a memory  
allocation

```
test.c:3:5: error: expected identifier or '(' before '[' token  
 3 | char[] get_a_string() {  
   |
```

# Recursion

- Recursion works in C too
- Keep in mind the stack

# Number of Coins

Write a program that:

- Accepts an amount of USD cents as a command-line positional arg
- Computes the min number of coins (pennies, nickels, dimes, quarters) needed to represent this amount
- ***NO loops***

```
$ ./a.out 15
Minimum coins needed: 2
$ ./a.out 51
Minimum coins needed: 3
$ ./a.out 104
Minimum coins needed: 8
$ ./a.out 2
Minimum coins needed: 2
```

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

int change(int amount) {
    // Base cases
    if (amount == 1 || amount == 5 || amount == 10 || amount == 25) {
        return 1;
    }
    if (amount > 25) {
        return 1 + change(amount - 25); // Quarter
    } else if (amount > 10) {
        return 1 + change(amount - 10); // Dime
    } else if (amount > 5) {
        return 1 + change(amount - 5); // Nickel
    } else {
        return 1 + change(amount - 1); // Penny
    }
}

int main(int argc, char** argv) {
    int x = atoi(argv[1]);
    int number_of_coins = change(x);
    printf("Minimum coins needed: %d\n", number_of_coins);
    return 0;
}
```



```

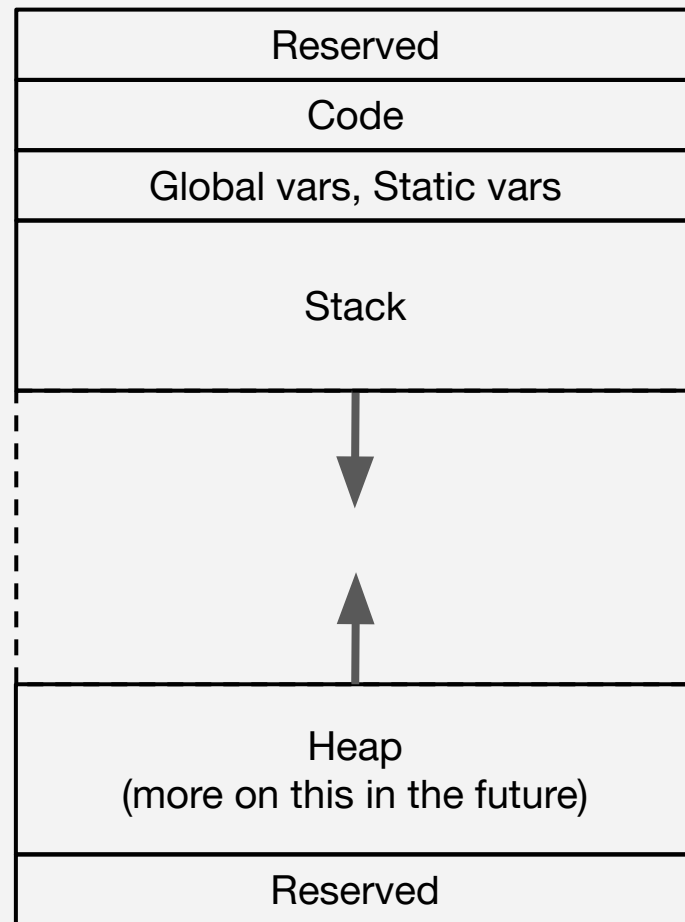
#include <stdio.h>
#include <stdlib.h>

int change(int amount) {
    // Base cases
    if (amount == 1 || amount == 5 || amount == 10 || amount == 25) {
        return 1;
    }
    if (amount > 25) {
        return 1 + change(amount - 25); // Quarter
    } else if (amount > 10) {
        return 1 + change(amount - 10); // Dime
    } else if (amount > 5) {
        return 1 + change(amount - 5); // Nickel
    } else {
        return 1 + change(amount - 1); // Penny
    }
}

int main(int argc, char** argv) {
    int x = atoi(argv[1]);
    int number_of_coins = change(x);
    printf("Minimum coins needed: %d\n", number_of_coins);
    return 0;
}

```

## Program Memory Layout



Low addr

High addr

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

```
int change(int amount) {
    // Base cases
    if (amount == 1 || amount == 5 || amount == 10 || amount == 25) {
        return 1;
    }
    if (amount > 25) {
        return 1 + change(amount - 25); // Quarter
    } else if (amount > 10) {
        return 1 + change(amount - 10); // Dime
    } else if (amount > 5) {
        return 1 + change(amount - 5); // Nickel
    } else {
        return 1 + change(amount - 1); // Penny
    }
}
```

```
int main(int argc, char** argv) {
    int x = atoi(argv[1]);
    int number_of_coins = change(x);
    printf("Minimum coins needed: %d\n", number_of_coins);
    return 0;
}
```

## Stack Example

main's caller's stack frame

main's stack frame

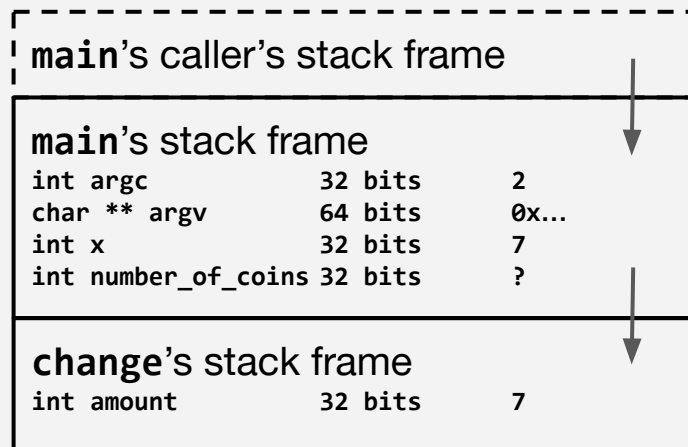
int argc	32 bits	2
char ** argv	64 bits	0x...
int x	32 bits	7
int number_of_coins	32 bits	?

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

```
int change(int amount) {
    // Base cases
    if (amount == 1 || amount == 5 || amount == 10 || amount == 25) {
        return 1;
    }
    if (amount > 25) {
        return 1 + change(amount - 25); // Quarter
    } else if (amount > 10) {
        return 1 + change(amount - 10); // Dime
    } else if (amount > 5) {
        return 1 + change(amount - 5); // Nickel
    } else {
        return 1 + change(amount - 1); // Penny
    }
}
```

```
int main(int argc, char** argv) {
    int x = atoi(argv[1]);
    int number_of_coins = change(x);
    printf("Minimum coins needed: %d\n", number_of_coins);
    return 0;
}
```

## Stack Example

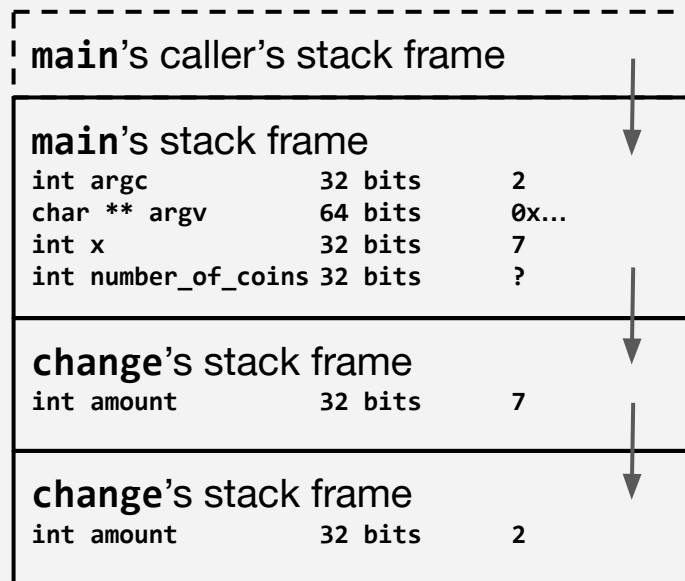


```
#include <stdio.h>
#include <stdlib.h>
```

```
int change(int amount) {
    // Base cases
    if (amount == 1 || amount == 5 || amount == 10 || amount == 25) {
        return 1;
    }
    if (amount > 25) {
        return 1 + change(amount - 25); // Quarter
    } else if (amount > 10) {
        return 1 + change(amount - 10); // Dime
    } else if (amount > 5) {
        return 1 + change(amount - 5); // Nickel
    } else {
        return 1 + change(amount - 1); // Penny
    }
}
```

```
int main(int argc, char** argv) {
    int x = atoi(argv[1]);
    int number_of_coins = change(x);
    printf("Minimum coins needed: %d\n", number_of_coins);
    return 0;
}
```

## Stack Example



```

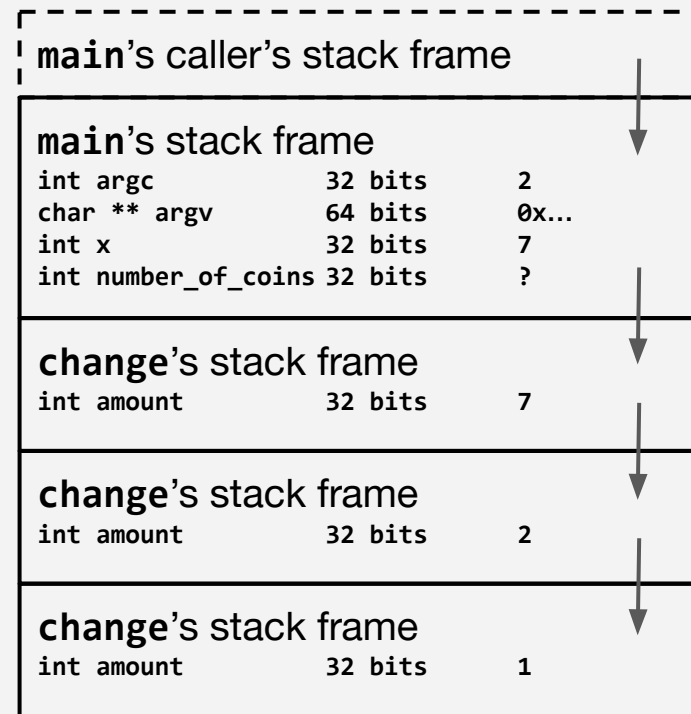
#include <stdio.h>
#include <stdlib.h>

int change(int amount) {
    // Base cases
    if (amount == 1 || amount == 5 || amount == 10 || amount == 25) {
        return 1;
    }
    if (amount > 25) {
        return 1 + change(amount - 25); // Quarter
    } else if (amount > 10) {
        return 1 + change(amount - 10); // Dime
    } else if (amount > 5) {
        return 1 + change(amount - 5); // Nickel
    } else {
        return 1 + change(amount - 1); // Penny
    }
}

int main(int argc, char** argv) {
    int x = atoi(argv[1]);
    int number_of_coins = change(x);
    printf("Minimum coins needed: %d\n", number_of_coins);
    return 0;
}

```

## Stack Example

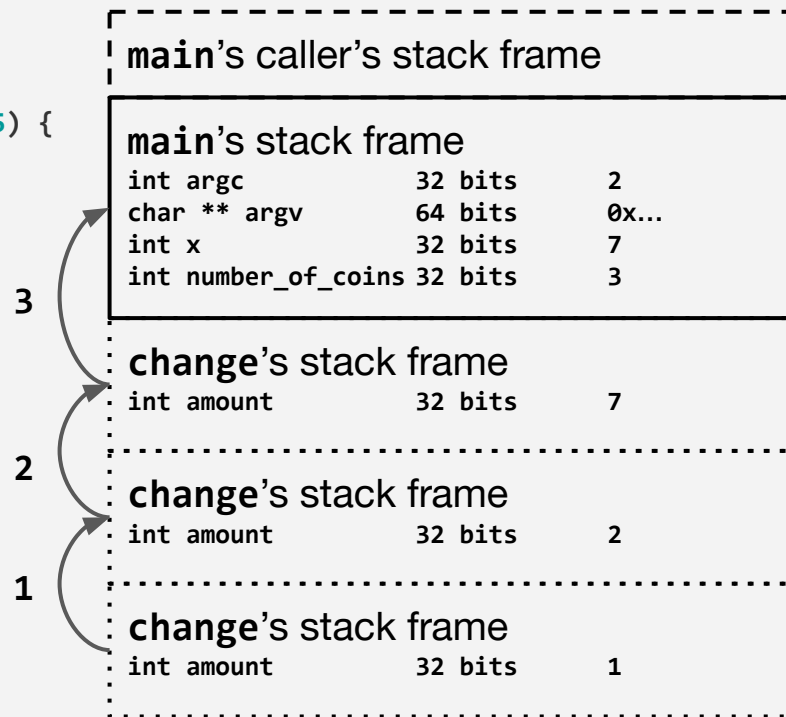


```
#include <stdio.h>
#include <stdlib.h>
```

```
int change(int amount) {
    // Base cases
    if (amount == 1 || amount == 5 || amount == 10 || amount == 25) {
        return 1;
    }
    if (amount > 25) {
        return 1 + change(amount - 25); // Quarter
    } else if (amount > 10) {
        return 1 + change(amount - 10); // Dime
    } else if (amount > 5) {
        return 1 + change(amount - 5); // Nickel
    } else {
        return 1 + change(amount - 1); // Penny
    }
}
```

```
int main(int argc, char** argv) {
    int x = atoi(argv[1]);
    int number_of_coins = change(x);
    printf("Minimum coins needed: %d\n", number_of_coins);
    return 0;
}
```

## Stack Example

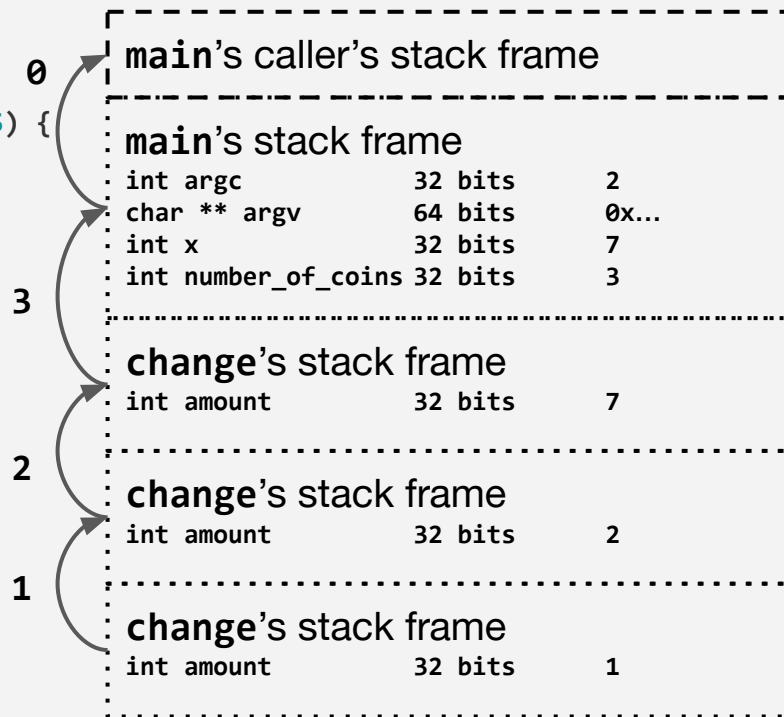


```
#include <stdio.h>
#include <stdlib.h>
```

```
int change(int amount) {
    // Base cases
    if (amount == 1 || amount == 5 || amount == 10 || amount == 25) {
        return 1;
    }
    if (amount > 25) {
        return 1 + change(amount - 25); // Quarter
    } else if (amount > 10) {
        return 1 + change(amount - 10); // Dime
    } else if (amount > 5) {
        return 1 + change(amount - 5); // Nickel
    } else {
        return 1 + change(amount - 1); // Penny
    }
}
```

```
int main(int argc, char** argv) {
    int x = atoi(argv[1]);
    int number_of_coins = change(x);
    printf("Minimum coins needed: %d\n", number_of_coins);
    return 0;
}
```

## Stack Example



# .h and .c files

- For large code projects in C, code is organized into .c and .h files
  - **Dot C** files: Where the *implementation* live
  - **Dot H** files: Where the *declarations / prototypes* live
- For libraries, put the code in .c and the function prototypes in .h
- In one of the future PAs you'll be tasked with writing a library as a part of it



# printing.h

# printing.c

# test.c

```
/**  
 * print an integer  
 */  
void print_int(int x);  
  
/**  
 * print an float  
 */  
void print_float(float x);  
  
. . .
```

```
#include "printing.h"  
#include <stdio.h>  
  
void print_int(int x) {  
    printf("%d\n", x);  
}  
  
void print_float(float x) {  
    printf("%f\n", x);  
}  
  
. . .
```

```
#include "printing.h"  
#include <stdio.h>  
  
int main() {  
    int x = 100;  
    float y = 100.128317281;  
    printf("Numbers!")  
    print_int(x);  
    print_float(y);  
}
```

```
$ gcc -Wall -Werror -std=c11 -c printing.c
```

```
$ gcc -Wall -Werror -std=c11 -o test test.c printing.o
```

# C standard library header files

- A library that you can assume should be installed and available on a UNIX system that supports C (most systems!)
- [https://en.wikipedia.org/wiki/C\\_standard\\_library#Header\\_files](https://en.wikipedia.org/wiki/C_standard_library#Header_files)

# The C Standard Library

In this course, you are allowed to use any function(s) from the C standard library, unless specifically restricted

# The C Standard Library

What headers are in the C standard library?

[https://en.wikipedia.org/wiki/C\\_standard\\_library](https://en.wikipedia.org/wiki/C_standard_library)

# The C Standard Library

Where does C search for these header files?

```
$ /usr/lib/gcc/x86_64-linux-gnu/9/cc1 -v
```

# The C Standard Library

How to search for where specifically one is?

```
$ ls /usr/include | grep HEADER_NAME.h
```

# The C Standard Library

After you've looked through it and found a function you want to use, how do you get more information about it?

```
$ man function_name
```

# Can also try:

```
$ locate stdbool.h
```

```
$ echo '#include <stdbool.h>' | cpp -H -o /dev/null 2>&1 | head -n1
```

<https://stackoverflow.com/questions/13079650/how-can-i-find-the-header-files-of-the-c-programming-language-in-linux>