C Bootcamp

CI Computer Girls

April 29, 2016

Hello World

```c
#include <stdio.h>

main() {
  printf("Hello, world!\n");
}
```

- A C program consists of *functions* and *variables*.

Hello World

```c
#include <stdio.h>

main() {
  printf("Hello, world!\n");
}
```

- A C program consists of *functions* and *variables*.
- A function contains *statements* that specify the computing operations to be done.

Hello World

```c
#include <stdio.h>

main() {
  printf("Hello, world!\n");
}
```

- A C program consists of *functions* and *variables*.
- A function contains *statements* that specify the computing operations to be done.
- Variables store values to be used during computation.

Hello World

```c
#include <stdio.h>

main() {
  printf("Hello, world!\n");
}
```

- A C program consists of *functions* and *variables*.
- A function contains *statements* that specify the computing operations to be done.
- Variables store values to be used during computation.
- Normally you can name functions whatever you like, but every program must contain a function named `main`.

Hello World

```c
#include <stdio.h>

main() {
  printf("Hello, world!\n");
}
```

- In this example `printf` is a function that takes a *character string* as its argument.

Hello World

```c
#include <stdio.h>

main() {
  printf("Hello, world!\n");
}
```

- In this example `printf` is a function that takes a *character string* as its argument.
- Copy the code above into an empty file `hello.c` in your `task1` directory (We'll help you find it.), and then from your terminal:

Hello World

```c
#include <stdio.h>

main() {
  printf("Hello, world!\n");
}
```

- In this example `printf` is a function that takes a *character string* as its argument.
- Copy the code above into an empty file `hello.c` in your `task1` directory (We'll help you find it.), and then from your terminal:

```
# cd ~/Desktop/bootcamp/task1
# gcc hello.c
# ./a.out
```

Prompts

From your terminal,

```
# cd ../task2
```

then open the file `prompt.c` in your text editor. You should see the following:

Prompts

From your terminal,

```
# cd ../task2
```

then open the file `prompt.c` in your text editor. You should see the following:

```c
#include <stdio.h>

main() {
  char name[40];
  printf("Enter your name:\n");

  // YOUR TASK: Prompt the user for their name say hello.

}
```

Prompts

```c
#include <stdio.h>

main() {
  char name[40];
  printf("Enter your name:\n");

  // YOUR TASK: Prompt the user for their name say hello.

}
```

- For this task, we'll make use of a new function

```c
scanf(char* format, ...)
```

Prompts

```
#include <stdio.h>

main() {
  char name[40];
  printf("Enter your name:\n");

  // YOUR TASK: Prompt the user for their name say hello.

}
```

- For this task, we'll make use of a new function

```
scanf(char* format, ...)
```

- `scanf` reads characters from your terminal, interprets them according to the `format` you provide (consult your cheatsheet), and stores the results in the remaining arguments.

Prompts

```c
#include <stdio.h>

main() {
  char name[40];
  printf("Enter your name:\n");

  // YOUR TASK: Prompt the user for their name say hello.

}
```

- For this task, we'll make use of a new function

```c
scanf(char* format, ...)
```

- `scanf` reads characters from your terminal, interprets them according to the `format` you provide (consult your cheatsheet), and stores the results in the remaining arguments.
- For example, to store a user-given string in `name`,

```c
scanf("%s", name);
```

Prompts

```c
#include <stdio.h>

main() {
  char name[40];
  printf("Enter your name:\n");

  // YOUR TASK: Prompt the user for their name say hello.

}
```

- For example, to store a user-given string in `name`,

  ```c
  scanf("%s", name);
  ```

- Similarly, `printf` can be given format specifiers in its first argument and will print the rest of its arguments accordingly.

  ```c
  printf("Goodbye %s", name);
  ```

Prompts

```c
#include <stdio.h>

main() {
  char name[40];
  printf("Enter your name:\n");

  // YOUR TASK: Prompt the user for their name say hello.

}
```

- For example, to store a user-given string in `name`,
  ```c
  scanf("%s", name);
  ```
- Similarly, `printf` can be given format specifiers in its first argument and will print the rest of its arguments accordingly.
  ```c
  printf("Goodbye %s", name);
  ```
- Complete your task (Ask for your help if you're stuck!), and run your program.

Arguments

From your terminal,

```
# cd ../task3
```

then open the file `arguments.c` in your text editor. You should see the following:

Arguments

From your terminal,

```
# cd ../task3
```

then open the file `arguments.c` in your text editor. You should see the following:

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

int main(int argc, char* argv[]) {
  if (argc < 3) {
    printf("Usage: %s <name> <integer>\n", argv[0]);
    return -1;
  }

  // YOUR TASK: Read the user’s name and an integer
  // from command line arguments, then say hello
  // to the user as many times as given by the integer.

}
```

Arguments

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

- Note that our `main` has grown a little.

Arguments

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

- Note that our `main` has grown a little.
- The first `int` tells us that this function will return an integer.

Arguments

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

- Note that our `main` has grown a little.
- The first `int` tells us that this function will return an integer.
- `int argc` and `char* argv[]` are parameters to `main`.

Arguments

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

- Note that our `main` has grown a little.
- The first `int` tells us that this function will return an integer.
- `int argc` and `char* argv[]` are parameters to `main`.
  - `char* argv[]` is an array of strings containing all the arguments we'll pass when we run our program. (More on that later.)

Arguments

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

- Note that our `main` has grown a little.
- The first `int` tells us that this function will return an integer.
- `int argc` and `char* argv[]` are parameters to `main`.
    - `char* argv[]` is an array of strings containing all the arguments we'll pass when we run our program. (More on that later.)
    - `int argc` is an integer indicating the length of `argv` or the number of strings contained within.

Arguments

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

For example, if we invoke our program as follows:

```
# ./a.out CiComputerGirls 5
```

Arguments

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

For example, if we invoke our program as follows:

```
# ./a.out CiComputerGirls 5
```

- Then `argc` contains the integer 3.

Arguments

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

For example, if we invoke our program as follows:

```
# ./a.out CiComputerGirls 5
```

- Then `argc` contains the integer 3.
- `argv[0]` contains the string `"a.out"`.

Arguments

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

For example, if we invoke our program as follows:

```
# ./a.out CiComputerGirls 5
```

- Then `argc` contains the integer 3.
- `argv[0]` contains the string `"a.out"`.
- `argv[1]` contains the string `"CiComputerGirls"`.

Arguments

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

For example, if we invoke our program as follows:

```
# ./a.out CiComputerGirls 5
```

- Then `argc` contains the integer 3.
- `argv[0]` contains the string `"a.out"`.
- `argv[1]` contains the string `"CiComputerGirls"`.
- `argv[2]` contains the string `"5"`.

Arguments

```c
int main(int argc, char* argv[]) {
  if (argc < 3) {
    printf("Usage: %s <name> <integer>\n", argv[0]);
    exit(-1);
  }
  ...
}
```

- In the given code, we examine `argc` in the condition of our if-statement to ensure our program was passed the correct number of arguments.

Arguments

```c
int main(int argc, char* argv[]) {
  if (argc < 3) {
    printf("Usage: %s <name> <integer>\n", argv[0]);
    exit(-1);
  }
  ...
}
```

- In the given code, we examine `argc` in the condition of our if-statement to ensure our program was passed the correct number of arguments.
- And if not, we print a helpful message and exit with an error code.

Arguments

```c
int main(int argc, char* argv[]) {
  if (argc < 3) {
    printf("Usage: %s <name> <integer>\n", argv[0]);
    exit(-1);
  }
  ...
}
```

- In the given code, we examine `argc` in the condition of our if-statement to ensure our program was passed the correct number of arguments.
- And if not, we print a helpful message and exit with an error code.
- Note that our helpful message prints the value of `argv[0]`. The first string in `argv` will always be the name of your program.

Arguments

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

  // YOUR TASK: Read the user's name and an integer
  // from command line arguments, then say hello
  // to the user as many times as given by the integer.

}
```

To complete your task,

- Use the function `atoi` to convert the value of `argv[2]` into an `int`.

  ```c
  int atoi(char* s)
  ```

Arguments

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

  // YOUR TASK: Read the user's name and an integer
  // from command line arguments, then say hello
  // to the user as many times as given by the integer.

}
```

To complete your task,

- Use the function `atoi` to convert the value of `argv[2]` into an `int`.
  ```c
  int atoi(char* s)
  ```
- `atoi` converts the string `s` into an `int`. For example,
  ```c
  int five = atoi("5");
  ```

Arguments

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

  // YOUR TASK: Read the user's name and an integer
  // from command line arguments, then say hello
  // to the user as many times as given by the integer.

}
```

To complete your task,

- Use the function `atoi` to convert the value of `argv[2]` into an `int`.
  ```c
  int atoi(char* s)
  ```
- `atoi` converts the string `s` into an `int`. For example,
  ```c
  int five = atoi("5");
  ```
- Then use a for- or while-loop to print your message as many times as needed.

Pointers

Arrays

Structs

Headers