

C Bootcamp

CI Computer Girls

April 29, 2016

Hello World

```
1  #include <stdio.h>
2
3  main() {
4      printf("Hello, world!\n");
5  }
```

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

Hello World

```
1  #include <stdio.h>
2
3  main() {
4      printf("Hello, world!\n");
5  }
```

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

Hello World

```
1  #include <stdio.h>
2
3  main() {
4      printf("Hello, world!\n");
5  }
```

- 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

```
1  #include <stdio.h>
2
3  main() {
4      printf("Hello, world!\n");
5  }
```

- 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

```
1  #include <stdio.h>
2
3  main() {
4      printf("Hello, world!\n");
5  }
```

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

Hello World

```
1  #include <stdio.h>
2
3  main() {
4      printf("Hello, world!\n");
5  }
```

- 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

```
1  #include <stdio.h>
2
3  main() {
4      printf("Hello, world!\n");
5  }
```

- 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:

```
1  #include <stdio.h>
2
3  main() {
4      char name[40];
5      printf("Enter your name:\n");
6
7      // YOUR TASK: Prompt the user for their name say hello.
8
9  }
```

Prompts

```
1  #include <stdio.h>
2
3  main() {
4      char name[40];
5      printf("Enter your name:\n");
6
7      // YOUR TASK: Prompt the user for their name say hello.
8
9  }
```

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

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

Prompts

```
1  #include <stdio.h>
2
3  main() {
4      char name[40];
5      printf("Enter your name:\n");
6
7      // YOUR TASK: Prompt the user for their name say hello.
8
9  }
```

- 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

```
1  #include <stdio.h>
2
3  main() {
4      char name[40];
5      printf("Enter your name:\n");
6
7      // YOUR TASK: Prompt the user for their name say hello.
8
9  }
```

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

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

Prompts

```
1  #include <stdio.h>
2
3  main() {
4      char name[40];
5      printf("Enter your name:\n");
6
7      // YOUR TASK: Prompt the user for their name say hello.
8
9  }
```

- For example, to store a user-given string in `name`,
`scanf("%s", name);`
- Similarly, `printf` can be given format specifiers in its first argument and will print the rest of its arguments accordingly.
`printf("Goodbye %s", name);`

Prompts

```
1  #include <stdio.h>
2
3  main() {
4      char name[40];
5      printf("Enter your name:\n");
6
7      // YOUR TASK: Prompt the user for their name say hello.
8
9  }
```

- For example, to store a user-given string in `name`,
`scanf("%s", name);`
- Similarly, `printf` can be given format specifiers in its first argument and will print the rest of its arguments accordingly.
`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:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char* argv[]) {
5      if (argc < 3) {
6          printf("Usage: %s <name> <integer>\n", argv[0]);
7          return -1;
8      }
9
10     // YOUR TASK: Read the user's name and an integer
11     // from command line arguments, then say hello
12     // to the user as many times as given by the integer.
13
14 }
```

Arguments

```
1  int main(int argc, char* argv[]) {  
2      ...  
3  }
```

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

Arguments

```
1 int main(int argc, char* argv[]) {  
2     ...  
3 }
```

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

Arguments

```
1  int main(int argc, char* argv[]) {  
2      ...  
3  }
```

- 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

```
1 int main(int argc, char* argv[]) {  
2     ...  
3 }
```

- 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

```
1 int main(int argc, char* argv[]) {  
2     ...  
3 }
```

- 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

```
1  int main(int argc, char* argv[]) {  
2      ...  
3  }
```

For example, if we invoke our program as follows:

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

Arguments

```
1 int main(int argc, char* argv[]) {  
2     ...  
3 }
```

For example, if we invoke our program as follows:

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

- Then `argc` contains the integer 3.

Arguments

```
1  int main(int argc, char* argv[]) {  
2      ...  
3  }
```

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

```
1  int main(int argc, char* argv[]) {  
2      ...  
3  }
```

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

```
1 int main(int argc, char* argv[]) {  
2     ...  
3 }
```

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

```
1  int main(int argc, char* argv[]) {  
2      if (argc < 3) {  
3          printf("Usage: %s <name> <integer>\n", argv[0]);  
4          exit(-1);  
5      }  
6      ...  
7  }
```

- 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

```
1  int main(int argc, char* argv[]) {  
2      if (argc < 3) {  
3          printf("Usage: %s <name> <integer>\n", argv[0]);  
4          exit(-1);  
5      }  
6      ...  
7  }
```

- 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

```
1  int main(int argc, char* argv[]) {  
2      if (argc < 3) {  
3          printf("Usage: %s <name> <integer>\n", argv[0]);  
4          exit(-1);  
5      }  
6      ...  
7  }
```

- 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

```
1  int main(int argc, char* argv[]) {  
2      ...  
3  
4      // YOUR TASK: Read the user's name and an integer  
5      // from command line arguments, then say hello  
6      // to the user as many times as given by the integer.  
7  
8  }
```

To complete your task,

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

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

Arguments

```
1  int main(int argc, char* argv[]) {  
2      ...  
3  
4      // YOUR TASK: Read the user's name and an integer  
5      // from command line arguments, then say hello  
6      // to the user as many times as given by the integer.  
7  
8  }
```

To complete your task,

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

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

- `atoi` converts the string `s` into an `int`. For example,

```
int five = atoi("5");
```


Arguments

```
1  int main(int argc, char* argv[]) {  
2      ...  
3  
4      // YOUR TASK: Read the user's name and an integer  
5      // from command line arguments, then say hello  
6      // to the user as many times as given by the integer.  
7  
8  }
```

To complete your task,

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

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

- `atoi` converts the string `s` into an `int`. For example,

```
int five = atoi("5");
```

- Then use a for- or while-loop to print your message as many times as needed.

Pointers

Now `cd` into the `task3` directory and open `pointers.c` in your text editor.

Pointers

Now `cd` into the `task3` directory and open `pointers.c` in your text editor.

```
1  #include <stdio.h>
2
3  main() {
4      char* string1 = "This is the first string.";
5      char* string2 = "This is the second string.";
6
7      // YOUR TASK: Write a function `swap` that swaps the pointers
8      // in its first and second arguments. Then invoke your function
9      // with string1 and string2.
10
11     printf("string1 = %s and string2 = %s\n", string1, string2);
12 }
```

Pointers

```
1 | main() {  
2 |     char *string1 = "This is the first string.";  
3 |     ...  
4 | }
```

- A *pointer* is a variable that contains the address in memory of another variable.

Pointers

```
1 | main() {  
2 |     char *string1 = "This is the first string."  
3 |     ...  
4 | }
```

- A *pointer* is a variable that contains the address in memory of another variable.
- C has two operators for dealing with pointers:

Pointers

```
1 | main() {  
2 |     char *string1 = "This is the first string.";  
3 |     ...  
4 | }
```

- A *pointer* is a variable that contains the address in memory of another variable.
- C has two operators for dealing with pointers:
 - `*` is the *dereferencing* operator. When applied to a pointer, it returns the value to which the pointer points.

Pointers

```
1 | main() {  
2 |     char *string1 = "This is the first string.";  
3 |     ...  
4 | }
```

- A *pointer* is a variable that contains the address in memory of another variable.
- C has two operators for dealing with pointers:
 - `*` is the *dereferencing* operator. When applied to a pointer, it returns the value to which the pointer points.
 - `&` does the opposite of `*`. When applied to a variable, it returns the address at which the variable is stored.

Pointers

```
1 | main() {  
2 |     char *string1 = "This is the first string.";  
3 |     ...  
4 | }
```

- Note that the above code declares `string1` to be a pointer. It states that `string1` points to the location in memory of the beginning of the assigned string.

Pointers

```
1 main() {  
2     char *string1 = "This is the first string.";  
3     ...  
4 }
```

- Note that the above code declares `string1` to be a pointer. It states that `string1` points to the location in memory of the beginning of the assigned string.
- Now note that `*string1` is a `char`. The `*` applies the dereferencing operator to `string1`, returning the first value at the location to which pointer points. In this case, `'T'`.

Pointers

To write your swap function, you must be aware of one more caveat:

- As in most languages, C passes arguments to functions by value, so there is no way for a called function to alter a variable in the calling function.

Pointers

To write your swap function, you must be aware of one more caveat:

- As in most languages, C passes arguments to functions by value, so there is no way for a called function to alter a variable in the calling function.
- Consider a function `swap` for swapping two integers:

```
1 void swap(int x, int y) { /* WRONG */  
2   int temp;  
3  
4   temp = x;  
5   x = y;  
6   y = temp;  
7 }
```

Pointers

To write your swap function, you must be aware of one more caveat:

- As in most languages, C passes arguments to functions by value, so there is no way for a called function to alter a variable in the calling function.
- Consider a function `swap` for swapping two integers:

```
1 void swap(int x, int y) { /* WRONG */  
2   int temp;  
3  
4   temp = x;  
5   x = y;  
6   y = temp;  
7 }
```

- Just like Java, this function will not work as intended.

Pointers

To write your swap function, you must be aware of one more caveat:

- As in most languages, C passes arguments to functions by value, so there is no way for a called function to alter a variable in the calling function.
- Consider a function `swap` for swapping two integers:

```
1 void swap(int x, int y) { /* WRONG */  
2   int temp;  
3  
4   temp = x;  
5   x = y;  
6   y = temp;  
7 }
```

- Just like Java, this function will not work as intended.
- `x` and `y` are swapped within the scope of our function `swap`, but since they were passed only by value to the function, they will not be swapped in any code that calls our `swap`.

Pointers

To obtain the desired effect, instead of writing a function that takes the values of the variables to be swapped, we write a function that takes the pointers to their values in memory:

Pointers

To obtain the desired effect, instead of writing a function that takes the values of the variables to be swapped, we write a function that takes the pointers to their values in memory:

```
1 void swap(int *px, int *py) {  
2     int temp;  
3  
4     temp = *px;    // temp gets the value to which px points.  
5     *px = *py;     // The value at px gets the value at py.  
6     *py = temp;    // The value to which py points gets temp.  
7 }
```

Pointers

To obtain the desired effect, instead of writing a function that takes the values of the variables to be swapped, we write a function that takes the pointers to their values in memory:

```
1 void swap(int *px, int *py) {  
2     int temp;  
3  
4     temp = *px;    // temp gets the value to which px points.  
5     *px = *py;     // The value at px gets the value at py.  
6     *py = temp;    // The value to which py points gets temp.  
7 }
```

We can invoke this `swap` like so:

```
1 int a = 5;  
2 int b = 10;  
3 swap(&a, &b);  
4 // Now a == 10, and b == 5.
```


Pointers

```
1 void swap(int *px, int *py) {  
2     int temp;  
3  
4     temp = *px;    // temp gets the value to which px points.  
5     *px = *py;     // The value at px gets the value at py.  
6     *py = temp;    // The value to which py points gets temp.  
7 }  
8  
9 int a = 5;  
10 int b = 10;  
11 swap(&a, &b);  
12 // Now a == 10, and b == 5.
```

- Note that to swap two `int` our function accepts two `int*`.

Pointers

```
1 void swap(int *px, int *py) {  
2     int temp;  
3  
4     temp = *px;    // temp gets the value to which px points.  
5     *px = *py;     // The value at px gets the value at py.  
6     *py = temp;    // The value to which py points gets temp.  
7 }  
8  
9 int a = 5;  
10 int b = 10;  
11 swap(&a, &b);  
12 // Now a == 10, and b == 5.
```

- Note that to swap two `int` our function accepts two `int*`.
- We use `&` to access the address at which our values are stored. In other words, to access *pointers* to our variables `a` and `b`.

Pointers

```
1  main() {  
2      char* string1 = "This is the first string.";  
3      char* string2 = "This is the second string.";  
4  
5      // YOUR TASK: Write a function `swap` that swaps the pointers  
6      // in its first and second arguments. Then invoke your function  
7      // with string1 and string2.  
8  
9      printf("string1 = %s and string2 = %s\n", string1, string2);  
10 }
```

- Since a string is already a pointer to a `char` (or a `char*`), your `swap` should take two pointers to `char` pointers (or two `char**`).

Pointers

```
1 main() {  
2     char* string1 = "This is the first string.";  
3     char* string2 = "This is the second string.";  
4  
5     // YOUR TASK: Write a function `swap` that swaps the pointers  
6     // in its first and second arguments. Then invoke your function  
7     // with string1 and string2.  
8  
9     printf("string1 = %s and string2 = %s\n", string1, string2);  
10 }
```

- Since a string is already a pointer to a `char` (or a `char*`), your `swap` should take two pointers to `char` pointers (or two `char**`).

Arrays

Structs

Headers