An Illustrative C program

C programs are composed of *functions.* Certain generic elements must also be present...

Comments

Note to self: comments remind you of what you thought you were doing.

```c
/* backwards
*   Display a text string backwards.
* 2015-08-17
*/
#include<stdio.h>   // for printf(), fgets()
#include<string.h>  // for strlen()

int main(int argc, char **argv)
{
    int i;             // a loop counter
    char *errmsg;      // pointer; used to mark errors
    char buffer[1000]; // 1000-character array of single characters, a.k.a. "textstring"

    printf("Enter text: "); // a prompting message
    errmsg = fgets(buffer, 1000, stdin);
    if (errmsg != NULL) {
        for (i = strlen(buffer) - 1; i >= 0; i--) {
            printf("%c ", buffer[i]);
        }
        printf("\n");
    }

    return 0;
}
//--------
```

one-line comments: C99 feature

A short C program

This program gets a text string, then prints it out backwards.

```c
/* backwards
*   Display a text string backwards.
* 2015-08-17
*/
#include<stdio.h>   // for printf(), fgets()
#include<string.h>  // for strlen()

int main(int argc, char **argv)
{
    int i;              // a loop counter
    char *errmsg;       // pointer; used to mark errors
    char buffer[1000];  // 1000-character array of single characters, a.k.a. "textstring"

    printf("Enter text: "); // a prompting message
    errmsg = fgets(buffer, 1000, stdin);
    if (errmsg != NULL) {
        for (i = strlen(buffer) - 1; i >= 0; i--) {
            printf("%c ", buffer[i]);
        }
        printf("\n");
    }

    return 0;
}
//--------
```

Let's examine the elements in this program:

Includes

#includes bring in standard definitions, so you don't have to write them yourself

You can also create your own include files.

```c
/* backwards
*   Display a text string backwards.
* 2015-08-17
*/
#include<stdio.h>   // for printf(), fgets()
#include<string.h>  // for strlen()

int main(int argc, char **argv)
{
    int i;              // a loop counter
    char *errmsg;       // pointer; used to mark errors
    char buffer[1000];  // 1000-character array of single characters, a.k.a. "textstring"

    printf("Enter text: "); // a prompting message
    errmsg = fgets(buffer, 1000, stdin);
    if (errmsg != NULL) {
        for (i = strlen(buffer) - 1; i >= 0; i--) {
            printf("%c ", buffer[i]);
        }
        printf("\n");
    }

    return 0;
}
//--------
```

Required In Every C Program

This is a *function*, named "main". It gets two arguments, named "argc" and "argv".

As it happens, this main( ) doesn't use its arguments.

```
/* backwards
*   Display a text string backwards.
* 2015-08-17
*/
#include<stdio.h>   // for printf(), fgets()
#include<string.h>  // for strlen()

int main(int argc, char **argv)
{
    int i;               // a loop counter
    char *errmsg;        // pointer; used to mark errors
    char buffer[1000];   // 1000-character array of single characters, a.k.a. "textstring"

    printf("Enter text: "); // a prompting message
    errmsg = fgets(buffer, 1000, stdin);
    if (errmsg != NULL) {
        for (i = strlen(buffer) - 1; i >= 0; i--) {
            printf("%c ", buffer[i]);
        }
        printf("\n");
    }

    return 0;
}
//--------
```

Variables

Variables must be declared before they can be used. Often at the beginning of the function they exist in (their *scope*).

```
/* backwards
*   Display a text string backwards.
* 2015-08-17
*/
#include<stdio.h>   // for printf(), fgets()
#include<string.h>  // for strlen()

int main(int argc, char **argv)
{
    int i;               // a loop counter
    char *errmsg;        // pointer; used to mark errors
    char buffer[1000];   // 1000-character array of single characters, a.k.a. "textstring"

    printf("Enter text: "); // a prompting message
    errmsg = fgets(buffer, 1000, stdin);
    if (errmsg != NULL) {
        for (i = strlen(buffer) - 1; i >= 0; i--) {
            printf("%c ", buffer[i]);
        }
        printf("\n");
    }

    return 0;
}
//--------
```

ANSI C: you *must* declare all variables *before* any other code.

C99: you *may* declare variables immediately before their first use (often a preferred style).

Output

The printf( ) function provides formatted output. Here are three minor uses of it.

```
/* backwards
*   Display a text string backwards.
* 2015-08-17
*/
#include<stdio.h>   // for printf(), fgets()
#include<string.h>  // for strlen()

int main(int argc, char **argv)
{
    int i;              // a loop counter
    char *errmsg;       // pointer; used to mark errors
    char buffer[1000];  // 1000-character array of single characters, a.k.a. "textstring"

    printf("Enter text: ");// a prompting message
    errmsg = fgets(buffer, 1000, stdin);
    if (errmsg != NULL) {
        for (i = strlen(buffer) - 1; i >= 0; i--) {
            printf("%c ", buffer[i]);
        }
        printf("\n");
    }

    return 0;
}
//--------
```

Input

The fgets( ) function obtains text input from a file-like object. "stdin" is the keyboard ("console").

It also returns a status value.

```
/* backwards
*   Display a text string backwards.
* 2015-08-17
*/
#include<stdio.h>   // for printf(), fgets()
#include<string.h>  // for strlen()

int main(int argc, char **argv)
{
    int i;              // a loop counter
    char *errmsg;       // pointer; used to mark errors
    char buffer[1000];  // 1000-character array of single characters, a.k.a. "textstring"

    printf("Enter text: "); // a prompting message
    errmsg = fgets(buffer, 1000, stdin);
    if (errmsg != NULL) {
        for (i = strlen(buffer) - 1; i >= 0; i--) {
            printf("%c ", buffer[i]);
        }
        printf("\n");
    }

    return 0;
}
//--------
```

If statements

This if( ) statement executes its body only if the condition is true. There is an optional "else" clause, which isn't used here.

The statement's body is a "for( )" loop followed by an output statement.

```c
/* backwards
*   Display a text string backwards.
* 2015-08-17
*/
#include<stdio.h>   // for printf(), fgets()
#include<string.h>  // for strlen()

int main(int argc, char **argv)
{
    int i;              // a loop counter
    char *errmsg;       // pointer; used to mark errors
    char buffer[1000];  // 1000-character array of single characters, a.k.a. "textstring"

    printf("Enter text: "); // a prompting message
    errmsg = fgets(buffer, 1000, stdin);
    if (errmsg != NULL) {
        for (i = strlen(buffer) - 1; i >= 0; i--) {
            printf("%c ", buffer[i]);
        }
        printf("\n");
    }

    return 0;
}
//--------
```

For Loops

This for( ) loop executes its body (an output statement) for a fixed number of times.

```c
/* backwards
*   Display a text string backwards.
* 2015-08-17
*/
#include<stdio.h>   // for printf(), fgets()
#include<string.h>  // for strlen()

int main(int argc, char **argv)
{
    int i;              // a loop counter
    char *errmsg;       // pointer; used to mark errors
    char buffer[1000];  // 1000-character array of single characters, a.k.a. "textstring"

    printf("Enter text: "); // a prompting message
    errmsg = fgets(buffer, 1000, stdin);
    if (errmsg != NULL) {
        for (i = strlen(buffer) - 1; i >= 0; i--) {
            printf("%c ", buffer[i]);
        }
        printf("\n");
    }

    return 0;
}
//--------
```

Returning a Value

```
/* backwards
*   Display a text string backwards.
* 2015-08-17
*/
#include<stdio.h>   // for printf(), fgets()
#include<string.h>  // for strlen()

int main(int argc, char **argv)
{
    int i;              // a loop counter
    char *errmsg;       // pointer; used to mark err
    char buffer[1000];  // 1000-character array of s

    printf("Enter text: "); // a prompting message
    errmsg = fgets(buffer, 1000, stdin);
    if (errmsg != NULL) {
        for (i = strlen(buffer) - 1; i >= 0; i--) {
            printf("%c ", buffer[i]);
        }
        printf("\n");
    }

    return 0;
}
//--------
```

The main( ) function is specified to return an integer, so it needs to end with a return statement. Here it just returns 0.

The operating system runs your program by calling main( ), and may do some cleanup depending on the returned value. 0 usually means "success".