

Week 5 Lecture 3

Compiler Errors

Agile Development

- Backlog of stories defines projects
 - Backlog contains all of the requirements currently known
- Stories define features of the project
 - Three elements: feature user, user's task, user's goal
- Examples are associated with stories to clarify
- Tests are associate with stories to make sure the feature works.

Example Backlog Item

- Story: As a user, I want to swap two integers, so I can try out a swap function.
- Example: I enter two numbers. The system displays the two numbers and the variable that contains them. When I hit return the system displays the variables and their contents, which are reversed.
- Test: I enter 1, then 2. The system displays “a=1; 2” followed by “a=2; b=1”

Story

- As a user, I want to swap two integers, so I can try out a swap function.
 - *As a user*: the user is a system user.
 - For such simple example it is hard to be realistic.
 - *I want to swap two integers*: the function is supposed to swap two integers.
 - *so I can sort items*: so I can try out a swap function.

Example

- I enter two numbers. The system displays the two numbers and the variable that contains them. When I hit return the system displays the variables and their contents, which are reversed.
 - For the programmer: clarifies the operation of the feature.
 - For the systems engineer: helps focus attention on the operation of the feature.
 - Helps raise questions about how the operation should work.
 - Frequently you will be both programmer and systems engineer.

Test

- I enter 1, then 2. The system displays “a=1; 2” then displays “a=2; b=1”.
 - Specifies a series of steps:
 - Create variables.
 - Call functions on the two variables.
 - Specifies an expected result
 - The values of the variables are changed.
 - Further clarifies the operation of the system for the programmer and system engineer.

Unit Test v System Test

- The tests associated with stories are system tests.
 - Systems tests check from the user's point of view.
 - They indicate that the system works properly.
- Unit tests are different
 - Unit tests check from the programmer's point of view
 - They indicate that invisible code works properly.
- Systems tests tell you whether the system works; Unit tests tell you why it is not working.

Algorithm

- Function Swap (a, b)
 - Requires local variable —————> Local variable temp;
 - Local variable holds value ↗ temp ← a;
 - Local value replaces lost value —————> a ← b;
 - Local value replaces lost value —————> b ← temp;

Developing a new program

- List what needs to happen.
- Create empty files (or files copied from elsewhere).
- Create Makefile.
- Create Test for first item.
- Run test to see it fail.
- Write code to make it succeed.

What needs to happen

- Get input: two integers
- Swap variables containing the inputs
- Print the outputs to see that the swap took place.

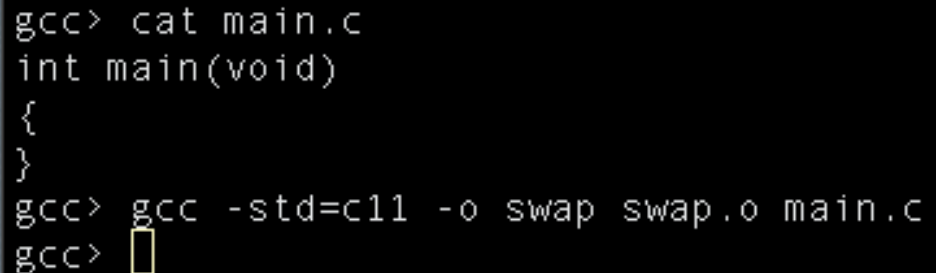
1. Create empty files and test

- Make three empty files with “touch”
 - main.c, swap.h swap.c
- Compile swap.c
 - Flags: use C 2011 standard, compile only, output swap.o
- Compile swap.o and main.c
 - Flags: use C 2011 standard, output swap,
- Error: Main not defined

```
gcc> touch main.c
gcc> touch swap.h
gcc> touch swap.c
gcc> gcc -std=c11 -c -o swap.o swap.c
gcc> gcc -std=c11 -o swap swap.o main.c
Undefined symbols for architecture x86_64:
  "_main", referenced from:
      implicit entry/start for main executable
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
gcc> █
```

2. Fix error: define main()

- Defined main function
- Test
- Test passes
 - In Unix nothing means no errors



```
gcc> cat main.c
int main(void)
{
}
gcc> gcc -std=c11 -o swap swap.o main.c
gcc> 
```

A terminal window with a black background and white text. It shows the definition of the `main` function in `main.c` and the compilation command `gcc -std=c11 -o swap swap.o main.c`. The prompt `gcc>` is shown at the end of the command line.

3. Add 2 lines of code

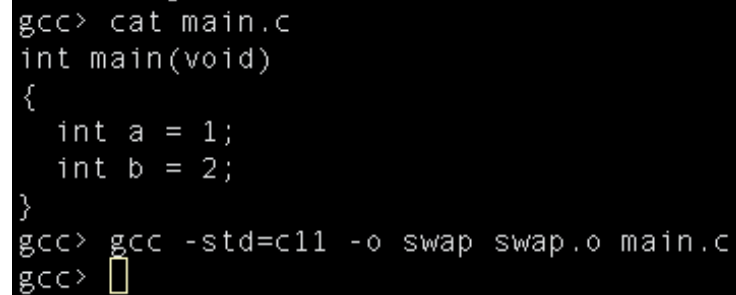
- Compilation fails
 - Missing semi-colon

```
gcc> cat main.c
int main(void)
{
    int a = 1;
    int b = 2
}
gcc> gcc -std=c11 -o swap swap.o main.c
main.c:4:12: error: expected ';' at end of declaration
    int b = 2
           ^
           ;
1 error generated.
gcc> 
```

4. Fix and test main.c

- Succeeds

- In Unix no errors means success



```
gcc> cat main.c
int main(void)
{
    int a = 1;
    int b = 2;
}
gcc> gcc -std=c11 -o swap swap.o main.c
gcc> 
```

A terminal window showing the compilation of main.c. The first command is 'cat main.c', which displays the contents of the file: 'int main(void) { int a = 1; int b = 2; }'. The second command is 'gcc -std=c11 -o swap swap.o main.c', which compiles the program. The prompt returns to 'gcc>' without any error messages.

- Notice that we do not need to recompile swap.c because we have not changed it.
 - We only need to compile when we change file.

5. Output values

- Print two ints, a and b, separated by comma: “%d, %d”
 - `printf(“%d, %d”, a, b)`
- Error: missing `<stdio.h>`
 - Contains definition of `printf`

```
gcc> cat main.c
int main(void)
{
    int a = 1;
    int b = 2;

    printf("%d, %d", a, b);
}
gcc> gcc -std=c11 -o swap swap.o main.c
main.c:6:3: warning: implicitly declaring library function 'printf' with type
      'int (const char *, ...)'
      printf("%d, %d", a, b);
      ^
main.c:6:3:      include the header <stdio.h> or explicitly provide a
      declaration for 'printf'
1 warning generated.
gcc> █
```

6. Fix compiler error

- Add:
 - `#include <stdio.h>`
- Compiles
- Runs
 - Because we added output, we can see it.
- Newline needed

```
gcc> cat main.c
#include <stdio.h>

int main(void)
{
    int a = 1;
    int b = 2;

    printf("%d, %d", a, b);
}
gcc> gcc -std=c11 -o swap swap.o main.c
gcc> ./swap
1, 2gcc> 
```


7. Fix Runtime error

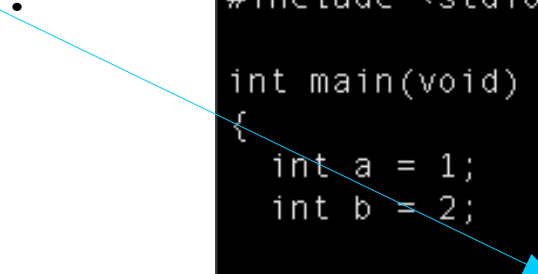
- Add newline '\n':

- Compiles
- Runs correctly

```
gcc> cat main.c
#include <stdio.h>

int main(void)
{
    int a = 1;
    int b = 2;

    printf("%d, %d\n", a, b);
}
gcc> gcc -std=c11 -o swap swap.o main.c
gcc> ./swap
1, 2
gcc> □
```



8. Refactor: name the output func

- Naming the printf makes it easier to understand what it does
- Also noticed that main was not returning a value
 - Added “return 0”

```
gcc> cat main.c
#include <stdio.h>

void print_vars(int a, int b)
{
    printf("%d, %d\n", a, b);
}

int main(void)
{
    int a = 1;
    int b = 2;

    print_vars(a, b);
    return 0;
}
gcc> gcc -std=c11 -o swap swap.o main.c
gcc> ./swap
1, 2
gcc> 
```

9. Refactor: Move func to swap.c

- Compiles swap.c with -c flag
- Error compiling main.c
 - Need header file swap.h to declare print_vars

```
gcc> cat swap.c
#include <stdio.h>

void print_vars(int a, int b)
{
    printf("%d, %d\n", a, b);
}
gcc> gcc -std=c11 -c -o swap.o swap.c
gcc> gcc -std=c11 -o swap swap.o main.c
main.c:8:3: warning: implicit declaration of function 'print_vars' is invalid in
      C99 [-Wimplicit-function-declaration]
    print_vars(a, b);
    ^
1 warning generated.
gcc> 
```

10. include swap.h in main.c

- File “swap.h”
 - Function declarations
- Included in main.c
 - Just like stdio.h
- Now program compiles and runs

```
gcc> cat swap.h
/*
 * swap.h
 */
void print_vars(int a, int b);
```

```
gcc> cat main.c
#include <stdio.h>
#include "swap.h"

int main(void)
{
    int a = 1;
    int b = 2;

    print_vars(a, b);
    return 0;
}
```

```
gcc> gcc -std=c11 -o swap swap.o main.c
gcc> ./swap
1, 2
```

11. Makefile

- Lets us write scripts to compile our programs
 - Target “swap” needs swap.o and main.c.
 - Target “swap.o” needs swap.c and swap.h
 - Both compile appropriately
 - Target “clean” removes file made
- We use the make program to:
 - Compile: “make” alone does first target
 - Clean up: “make clean”

```
gcc> cat makefile
swap: swap.o main.c
    gcc -std=c11 -o swap swap.o main.c

swap.o: swap.c swap.h
    gcc -std=c11 -c -o swap.o swap.c

clean:
    rm *.o swap
```

```
gcc> make clean
rm *.o swap
gcc> make
gcc -std=c11 -c -o swap.o swap.c
gcc -std=c11 -o swap swap.o main.c
gcc> █
```

Design for Testing

- The printf function can provide insight while you test.
 - Include printf statements to see what the program does
- Output is easy to observe.
 - Always write output routines first.

11. Makefile

- Lets us write scripts to compile our programs
 - Target “swap” needs swap.o and main.c.
 - Target “swap.o” needs swap.c and swap.h
 - Both compile appropriately
 - Target “clean” removes file made
- We use the make program to:
 - Compile: “make” alone does first target
 - Clean up: “make clean”

```
gcc> cat makefile
swap: swap.o main.c
    gcc -std=c11 -o swap swap.o main.c

swap.o: swap.c swap.h
    gcc -std=c11 -c -o swap.o swap.c

clean:
    rm *.o swap
```

```
gcc> make clean
rm *.o swap
gcc> make
gcc -std=c11 -c -o swap.o swap.c
gcc -std=c11 -o swap swap.o main.c
gcc> █
```

12. Swap first in main

- Uses standard algorithm
- We can use `print_vars` to see if it works
- Compile and run
 - Notice: `make` only compiles `swap`; `swap.o` already exists

```
gcc> cat main.c
#include <stdio.h>
#include "swap.h"

int main(void)
{
    int a = 1;
    int b = 2;
    int temp;

    print_vars(a, b);
    temp = a;
    a = b;
    b = temp;
    print_vars(a, b);
    return 0;
}
```

```
gcc> make
gcc -std=c11 -o swap swap.o main.c
gcc> ./swap
1, 2
2, 1
```


13. Refactor swap(a, b) to swap.c

```
gcc> cat swap.c
/*
 * swap.c
 */

#include <stdio.h>

void print_vars(int a, int b)
{
    printf("%d, %d\n", a, b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```

```
gcc> cat main.c
#include <stdio.h>
#include "swap.h"

int main(void)
{
    int a = 1;
    int b = 2;

    print_vars(a, b);
    swap(a, b);
    print_vars(a, b);
    return 0;
}
```

- Doesn't compile due to typo

```
gcc> make
gcc -std=c11 -c -o swap.o swap.c
gcc -std=c11 -o swap swap.o main.c
main.c:10:3: warning: implicit declaration of function 'swap' is invalid in C99
      [-Wimplicit-function-declaration]
    swap(a, b);
    ^
1 warning generated.
```

14. Refactor: fix typo

```
gcc> cat main.c
#include <stdio.h>
#include "swap.h"

int main(void)
{
    int a = 1;
    int b = 2;

    print_vars(a, b);
    swap(a, b);
    print_vars(a, b);
    return 0;
}
```

- Doesn't compile because swap not declared

```
gcc> cat swap.c
/*
 * swap.c
 */

#include <stdio.h>

void print_vars(int a, int b)
{
    printf("%d, %d\n", a, b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
    b = temp;
}
```

```
gcc> make
gcc -std=c11 -o swap swap.o main.o
main.c:10:3: warning: implicit declaration of function 'swap' is invalid in C99
      [-Wimplicit-function-declaration]
    swap(a, b);
    ^
1 warning generated.
```

15. Refactor: add declaration

```
gcc> cat main.c
#include <stdio.h>
#include "swap.h"

int main(void)
{
    int a = 1;
    int b = 2;

    print_vars(a, b);
    swap(a, b);
    print_vars(a, b);
    return 0;
}
```

```
gcc> cat swap.h
/*
 * swap.h
 */
void print_vars(int a, int b);
void swap(int a, int b);
```

```
gcc> cat swap.c
/*
 * swap.c
 */
#include <stdio.h>

void print_vars(int a, int b)
{
    printf("%d, %d\n", a, b);
}

void swap(int a, int b)
{
    int temp;

    temp = a;
    a = b;
```

- Doesn't compile but doesn't work.

```
gcc> make
gcc -std=c11 -c -o swap.o swap.c
gcc -std=c11 -o swap swap.o main.c
gcc> ./swap
1, 2
1, 2
```

16. Refactor: call by reference

```
gcc> cat main.c
#include <stdio.h>
#include "swap.h"

int main(void)
{
    int a = 1;
    int b = 2;

    print_vars(a, b);
    swap(&a, &b);
    print_vars(a, b);
    return 0;
}
```

```
gcc> cat swap.h
/*
 * swap.h
 */
void print_vars(int a, int b);
void swap(int *a, int *b);
```

```
gcc> cat swap.c
/*
 * swap.c
 */
#include <stdio.h>

void print_vars(int a, int b)
{
    printf("%d, %d\n", a, b);
}

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

    temp = *a;
    *a = *b;
    *b = temp;
}
```

- Compiles and works.

```
gcc> make
gcc -std=c11 -c -o swap.o swap.c
gcc -std=c11 -o swap swap.o main.c
gcc> ./swap
1, 2
2, 1
```

17. Add input

- Add to main.

```
gcc> cat main.c
#include <stdio.h>
#include "swap.h"

int main(void)
{
    int a = 1;
    int b = 2;

    printf ("Input a> ");
    scanf("%d", &a);
    printf ("Input a> ");
    scanf("%d", &b);
    print_vars(a, b);
    swap(&a, &b);
    print_vars(a, b);
    return 0;
}
```

- Compiles and works.

```
gcc> make
gcc -std=c11 -o swap swap.o main.c
gcc> ./swap
Input a> 3
Input a> 4
3, 4
4, 3
```

18. Refactor: move to function

```
gcc> cat main.c
#include <stdio.h>
#include "swap.h"

int main(void)
{
    int a = 1;
    int b = 2;

    input_vars(&a, &b);
    print_vars(a, b);
    swap(&a, &b);
    print_vars(a, b);
    return 0;
}
```

```
gcc> cat swap.h
/*
 * swap.h
 */
void print_vars(int a, int b);
void swap(int *a, int *b);
void input_vars(int *a, int *b);
```

```
gcc> cat swap.c
/*
 * swap.c
 */
#include <stdio.h>

void input_vars(int *a, int *b)
{
    printf ("Input a> ");
    scanf("%d", a);
    printf ("Input a> ");
    scanf("%d", b);
}

void print_vars(int a, int b)
{
    printf("%d, %d\n", a, b);
}

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

    temp = *a;
    *a = *b;
    *b = temp;
}
```

- Compiles and works.

```
gcc> make
gcc -std=c11 -c -o swap.o swap.c
gcc -std=c11 -o swap swap.o main.c
gcc> ./swap
Input a> 3
Input a> 4
3, 4
4, 3
```

Lesson

- Change as little as possible before compiling
- Fix all errors and warnings before compiling again
- Fix incorrect output before adding new features