

C Tutorial 1

CIT 593

1/31

Source files in C

C files have the extension `.c`

- There are can multiple `.c` files but only one file can contain the function `main()`
 - This is because `main()` is entry point of your program
- There is no restriction on the name of your source files in C
 - Unlike in java the filename has to be same as class name

CIT 593

2/31

Compiling C Programs

Use gcc compiler for

- Compiling and linking `.c` source files
- Example: `<prompt> gcc hello.c`
 - Produces a file `a.out` which contains the machine code (similar to `.obj`)
 - But `a.out` is not meaning full name
- Example: `<prompt> gcc -o Hello hello.c`
 - Option `-o` gives the (object) file with machine code
 - Now file called `Hello` contains machine code

CIT 593

3/31

Compiling C Programs (contd..)

Others options

- `gcc -S -o hello.s hello.c`
 - `hello.s` contains the assembly language code of the machine on which C program is compiled
 - Assembly code is translated in machine code

CIT 593

4/31

Running/Executing your C program

After the you get the machine code

- To execute/run your program in (esp. in bash) shell environment - type at the prompt
 - `./Hello` or `./a.out`
 - This option might be needed if you work remotely
- In some shell environment simply to do
 - `Hello` or `a.out`

CIT 593

5/31

Important

All C programs must compile & run on x86 machines

- Use eniac-I machines for your homeworks
- Login into machine:
 - <prompt> `ssh eniac-I.seas.upenn.edu`
- If you work from home use SecureCRT to login into eniac-I and use file Ftp to upload your source files

CIT 593

6/31

Example 1: circle.c

```
#define RADIUS 15.0
#include <stdio.h>
int main(){
    const float pi = 3.14159;
    float area;
    float circum;
    area = pi * RADIUS * RADIUS;
    circum = 2 * pi * RADIUS;
    printf("area = %f\n",area);
    printf("circum = %f\n",circum);
    return 0;
}
```

1. Compile
`gcc -o circle circle.c`
2. Run
`circle`
3. Print area & circum

CIT 593

7/31

Example 1: circle.c (contd..)

```
#define RADIUS 15.0
#include <stdio.h>
int main(){
    const float pi = 3.14159;
    float area;
    float circum;
    printf("area = %f\n",area);
    printf("circum = %f\n",circum);
    area = pi * RADIUS * RADIUS;
    ...
    return 0;
}
```

1. Print the value of area and circum before their assignment
2. Compile & Run
3. What do you see?
 - Some junk value
 - Initialization is very important.
 - Initialize area and circum to 0.0

CIT 593

8/31

Example 1: circle.c (contd..)

```
#define RADIUS 15.0
#include <stdio.h>
int main(){
    const float pi = 3.14159;
    float area = 0.0;
    /*float circum = 0.0*/;
    area = pi * RADIUS * RADIUS;
    circum = 2 * pi * RADIUS;
    printf("area = %f\n",area);
    printf("circum = %f\n",circum);
    return 0;
}
```

1. Comment out “double circum” either // or /* */
2. Compile & Run
3. C compiler comments ?

CIT 593

9/31

Example 1: circle.c – Undeclared Variable

Compiler complains:

circle.c: In function 'main':

circle.c:10: error: 'circum' undeclared (first use in this function)

circle.c:10: error: (Each undeclared identifier is reported only once

circle.c:10: error: for each function it appears in.)

Now fix the old error and remove a semicolon “;” from one of the statements and see what happens

CIT 593

10/31

Example 1: circle.c – Missing Semicolon

Compiler complains:

circle.c: In function 'main':

circle.c:8: error: expected ',' or ';' before 'float'

circle.c:13: error: 'area' undeclared (first use in this function)

circle.c:13: error: (Each undeclared identifier is reported only once

circle.c:13: error: for each function it appears in.)

Compiler complains error on a line before the actual line containing missing ‘;’

CIT 593

11/31

Example 1: circle.c (contd..)

```
#define RADIUS 15.0
```

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

```
int main(){
```

```
    const float pi = 3.14159;
```

```
    float area = 0.0;
```

```
    float circum = 0.0;
```

```
    area = pi * RADIUS * RADIUS;
```

```
    circum = 2 * pi * RADIUS;
```

```
    printf("area = %f\n",area);
```

```
    printf("circum = %f\n",circum);
```

```
}
```

1. Change program such that user is requested to enter RADIUS

2. How do we do that?
- scanf()

CIT 593

12/31

Example 1: circle.c using scanf()

```
/*#define RADIUS 15.0*/
#include <stdio.h>
int main(){
    const float pi = 3.14159;
    float area = 0.0;
    float circum = 0.0;
    float RADIUS = 0.0;
    printf("enter radius of circle\n");
    scanf("%f", &RADIUS);
    area = pi * RADIUS * RADIUS;
    circum = 2 * pi * RADIUS;
    printf("area = %e\n",area);
    printf("circum = %e\n",circum);
}
```

Limitations to scanf() ??

CIT 593 13/31

Example 1: circle.c – printf formatting (1)

```
/*#define RADIUS 15.0*/
#include <stdio.h>
int main(){
    const float pi = 3.14159;
    float area = 0.0;
    float circum = 0.0;
    float RADIUS = 0.0;
    printf("enter radius of circle\n");
    scanf("%f", &RADIUS);
    area = pi * RADIUS * RADIUS;
    circum = 2 * pi * RADIUS;
    printf("area = %.2f\n",area);
    printf("circum = %.2f\n",circum);
}
```

CIT 593 14/31

1. Compile & Run
2. What happens?
 - The precision changes
 - Default the precision is up to 6 places after decimal point

Example 1: circle.c – printf formatting (2)

```
/*#define RADIUS 15.0*/
#include <stdio.h>
int main(){
    const float pi = 3.14159;
    float area = 0.0;
    float circum = 0.0;
    float RADIUS = 0.0;
    printf("enter radius of circle\n");
    scanf("%f", &RADIUS);
    area = pi * RADIUS * RADIUS;
    circum = 2 * pi * RADIUS;
    printf("area = %15f\n",area);
    printf("circum = %15f\n",circum);
}
```

1. Compile & Run
2. What happens?
 - Output is printed many spaces from '='

CIT 593 15/31

Functions in C

Functions

- a.k.a subroutines
 - Piece of code that can be reused
 - In C each function
 - Must have return type. Example **int** foo()
 - If nothing is returned from that function then we put the keyword **void**. Example: **void** foo()
 - A function may or may not have arguments passed to it. Example: **int** foo(**int** a), **int** foo()
- CIT 593 16/31

Functions in C: Part 1

Functions can be written in the same file in the which function main() is declared

- Example: compare.c
- function can before or after the main(){}
 - Before: put the entire function before main()
 - Example: compare.c
 - After: declare function prototype before main()
 - Example: static.c

CIT 593

17/31

Example 2: compare.c

```
#include <stdio.h>
int compare(int x, int y){
    if (x > y){
        return 1;
    }
    else if(x < y){
        return 2;
    }
    else{
        return 0;
    }
}

int main(){
    int a = 0;
    int b = 0;
    int c = 0;
    printf("Enter a\n");
    scanf("%d",&a);

    printf("Enter b\n");
    scanf("%d",&b);
    c = compare(a,b);
    switch(c){
    case 0:
        printf("a and b are equal\n");
        break;
    case 1:
        printf("a is greater than b\n");
        break;
    case 2:
        printf("b is greater than a\n");
        break;
    default:
        printf("Can't Compare\n");
    }
}
```

CIT 593

18/31

Example 3: static.c

```
#include <stdio.h>
/*void showstat(int curr);*/
int main() {
    // int i;
    for (int i = 0; i < 5; i++)
    {
        showstat( i );
    }
}

void showstat( int curr ) {
    static int nStatic = 0;
    nStatic += curr;
    printf("nStatic = %d\n",nStatic);
}
```

1. Compile w/ function prototype commented out

2. Warning:
static.c:15: warning: conflicting types for 'showstat'
static.c:9: warning: previous implicit declaration of 'showstat' was here

3. Loop index must be declared outside the for loop statement

4. What does Static variable nStatic do?

CIT 593

19/31

Functions in C: Part 2

Functions also can be declared in separate files

- Usually done when programs are larger
- Different programmers work on their subparts

Functions in separate files should be accompanied by .h file

CIT 593

20/31

.h files

Purpose of header files:

- System header files declare the interfaces to parts of the operating system code
- This also applies to parts of user code that is being reused
 - Instead of typing the declaration every time just include the header file for interface
- To aid the compiler before hand in knowing the types of variable passed and return
- You can also define symbols, enums or data structures definitions that are going to be used in many places

System header files:

```
#include <filename.h>
```

User header files:

```
#include "filename.h"
```

CIT 593

21/31

Enumerations

Keyword **enum** declares a new type that take on symbolic values

- `enum colors { RED, GREEN, BLUE, YELLOW, MAUVE };`
- RED is now 0, GREEN is 1, etc.
- Gives meaning to constants, or groups constants
- Can start at value other than zero E.g. {RED = 1, ...}

```
enum colors house_color;
house_color = get_color(); //Note: colors house-color is illegal in C
switch (house_color) {
    case RED:
        /* code here */
        break;
    /* more here... */
}
```

Enums are just ints, but can provide more type checking

- C++ adds even more checking support

CIT 593

22/31

Example 5: month.c and month.h

month.h

- Contains enum definition called month

month.c

- Includes .h so that a variable of type month can be created

```
#include <stdio.h>
#include "month.h"
int main(){
    enum month m;
    printf("Enter month:");
    scanf("%d", &m);
    printf("month = %d\n", m);

    switch (m) {
        case APRIL : case JUNE : case
        SEP :case NOV:
            printf("Month has 30 days.\n");
            break;
        ....

        default: printf("Don't know that
        month.\n");
    }
}
```

CIT 593

23/31

Example 5: hypotenuse.h &hypotenuse.c

hypotenuse.h

- Contains the function's interface
- Example:
 - `float hypo(float , float);`
 - `stdio.h` contains function prototype of `printf()` and `scanf()` etc...

hypotenuse.c

- Contains declaration, along with the body of the function
 - i.e. function's implementation

CIT 593

24/31

Example 5: contd..

```
main.c
#include <stdio.h>
#include "hypotenuse.h"
```

```
int main(){
float side1 = 0.0;
float side2 = 0.0;
float hypo_side = 0.0;
printf("Enter side 1\n");
scanf("%f",&side1);
printf("Enter side 2\n");
scanf("%f",&side2);
```

```
hypo_side = hypo(side1, side2);
printf("Hypotenuse = %f\n",
hypo_side);
```

```
}
```

CIT 593

```
hypotenuse.c
#include <math.h>
#include "hypotenuse.h"
```

```
float hypo(float s1, float s2){
float a = s1;
float b = s2;
float h = 0.0;
```

```
a = a * a;
b = b * b;
```

```
h = sqrt(a + b);
return h;
}
```

Your own
function
included with ""

From <math.h>

double sqrt(double x)

In C no type checking,
compiler just converts
float to double

25/31

Compiling more than one .c file

To individually compile each file

- <prompt> gcc -c main.c
- <prompt> gcc -c hypotenuse.c
- -c option just compiles the files individually but does not link different files
- Creates file with extension .o Example: main.o
- Useful for debugging each file individually

To compile and link at same time:

- <prompt> gcc -o hypo main.c hypotenuse.c

CIT 593

26/31

Still getting an error???

Error:

```
/tmp/ccOxtexy.o: In function `hypo':
hypotenuse.c:(.text+0x50): undefined reference to `sqrt'
collect2: ld returned 1 exit status
```

Reason:

- add the `-lm` option to the end of the the `gcc` command line to force the linker to include the math library
- ```
<prompt> gcc -lm -o hypo main.c hypotenuse.c
```

CIT 593

27/31

## Static vs. Dynamic

C libraries are either *static* or *dynamic*

- Depending on how they are *linked* with an application program during compilation.

### Static library routines

- Copied into the executable file by the linker at compile time

### Dynamic library routines

- Not combined with the program at compile time;
- Instead, they are loaded into memory at *run* time when first called by the executing program

### By default

- The compiler uses the *dynamic* version of a library whenever possible.
  - The compiler can be forced to use static libraries via the use of the compiler flag called `-static`
- The static version of the C math library is called `libm.a` (.a is for "archive" in UNIX jargon).
  - The corresponding dynamic version of the C math library is called `libm.so` (.so is for "shared object" in UNIX jargon)

CIT 593

28/31

## Variable declaration with different gcc versions

On Linux gcc version is 4.1.2 variables don't need to be declared before they can be assigned a value.

- Example: `int counter = 10`
- Exception, in case of a for loop, the loop index needs to be declared before loop

On **Unix gcc version is 2.7.2**, variables needs to declared before assigning values. Example:

- `int counter;`  
`counter = 0;`  
You will get an error as stated below if you do `int counter = 10`  
`parse error before int`  
`'counter' undeclared (first use this function)`

CIT 593

29/31

## Differences with gcc versions

Another example: `//` vs. `/* */` for comments

**Bottom line:** things will change in the future, just have to learn to understand compiler comments or do some research (google is your friend)

To find gcc version for particular system:  
`<prompt> gcc -v`

CIT 593

30/31

## References

### Book References

- Kernighan, Ritchie: The C programming language, 2<sup>nd</sup> ed. Prentice Hall.
- Van der Linden: Expert C programming: Deep C Secrets. Prentice Hall.

Use FAQ page on website

Websites:

<http://www.cis.upenn.edu/~palsetia/technicalFAQ.html>

CIT 593

31/31