

## C Tutorial 1

CIT 593

C - 1

## 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 file name has to be same as class name)

CIT 593

C - 2

## 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 file with machine code
  - Now `hello` contains machine code

CIT 593

C - 3

## 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
- `gcc -o hello hello.s`
  - Assembly code is translated in machine code

CIT 593

C - 4

## Running/Executing your C program

After the you get the machine code

- Execute/run your program in bash shell environment by (we will do this for all hws)
  - <prompt> `./hello` or
  - <prompt> `./a.out`
- If in future you work on tcsh environment then
  - <prompt> `hello` or
  - <prompt> `a.out`

CIT 593

C - 5

## Important

All C programs must compile & run on x86 machines

- Use eniac-1 machines for your homeworks
- Login into machine:
  - <prompt> `ssh eniac-1.seas.upenn.edu`
- Use bash shell (so no need to change shell like we did in previous tutorials)
- If you work from home use SecureCRT to login into eniac-1 and use file Ftp to upload your source files

CIT 593

C - 6

## 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);
}
}CIT 593
```

1. Compile
2. Run
3. Print area & circum

C - 7

## 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;
    circum = 2 * pi * RADIUS;
    printf("area = %f\n",area);
    printf("circum = %f\n",circum);
}
}CIT 593
```

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

C - 8

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

CIT 593 C - 9

### 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

C - 10

### Example 1: circle.c – Missing Semicolon

#### Compiler complains:

circle.c: In function 'main':  
circle.c:6: error: expected ',' or ';' before 'float'  
circle.c:10: error: 'area' 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.)

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

CIT 593

C - 11

### 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 C - 12

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

CIT 593

C - 13

### 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

C - 14

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);
}
```

CIT 593

C - 15

1. Compile & Run

2. What happens?

- Output is printed many spaces from '='

### Example 1: printing as particular data type

```
/*#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 = %d\n",area);
    printf("circum = %d\n",circum);
}
```

CIT 593

C - 16

1. Compile & Run

2. What happens?

- area & circum are printed in scientific notation

## 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

C - 17

## Functions in C: Part 1

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

- Example: compare.c
- Can be written 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

C - 18

### 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

C - 19

### Example 3: static.c

```
#include <stdio.h>
/*void showstat(int curr);*/
int main() {
    int i;
    for (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:13: 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

C - 20

### Example 4: reverse.c for declaring arrays

```
#include <stdio.h>
#define MAXLINE 80
void reverse(char s[], int stringLength);
int getLine(char s[], int lim);
int main(){
    char charArray[MAXLINE];
    int charArraySize;
    int i;
    for(i = 0; i < MAXLINE; i++){
        charArray[i] = ' ';
    }
    printf("Enter a string\n");
    charArraySize = getLine(charArray, MAXLINE);
    printf("Before: %s\n", charArray);
    reverse(charArray, charArraySize);
    printf("After: %s\n", charArray);
}
```

- Creating character array o that can hold 1000 characters
- This is static array, as we have telling compiler to allocate memory for 1000 chars
- When we do pointers, we will learn dynamic creation

CIT 593

C - 21

### Example 4: reverse.c (contd..)

```
int getLine(char s[], int lim){
    int c;
    int i = 0;
    while(lim > 0 && (c = getchar()) !=
EOF && c != '\n')
    {
        if(c == '\n'){
            s[i] = '\0';
        }
        else{
            s[i] = c;
        }
        i++;
        lim--;
    }
    return i;
}

void reverse(char s[], int stringLength)
{
    char c;
    int i, j;

    for (i=0, j=stringLength-1; i < j; i++, j--)
    {
        c = s[i];
        s[i] = s [j];
        s[j] = c;
    }
}
```

CIT 593

C - 22

### 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

CIT 593

C - 23

### Example 5: hypotenuse.h &hypotenuse.c

#### hypotenuse.h

- .h files are called header files
- Contains the function prototype i.e.descriptions of functions and variables needed by the program
- To aid the compiler before hand in knowing the types of variable passed and return
- Example:
  - float hypo(float , float );
  - stdio.h contains function prototype of printf() and scanf() etc...

#### hypotenuse.c

- Contains signature and the body of the function
  - parameter(s) and return value type & how it is implemented

CIT 593

C - 24

## 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

C - 25

## 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

C - 26

## 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:

When including *math.h* in a program, you must add the *-lm* option to the end of the the *gcc* command line to force the linker to include the math library. Otherwise, the program will not compile.

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

CIT 593

C - 27

## Static vs. Dynamic libraries

C libraries are either *static* or *dynamic*, depending on how they are *linked* with an application program during compilation.

Static library routines

- are copied into the executable file by the linker at compile time.

Dynamic library routines

- are 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 *-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

C - 28

## Variable declaration with different versions of gcc

- On **Linux gcc version is 4.1.0**, 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. Example:  

```
int i;  
for(i = 0; i < 10; i++)
```
- 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 ddo `int counter = 10`  

```
parse error before int  
'counter' undeclared (first use this function)
```

CIT 593

C - 29

## gcc versions

Variable declaration is one example of the difference between old and new versions of gcc

- 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

C - 30

## File I/O in C

Other than reading and writing to screen, we can also read and write files

What you need:

- `stdio.h`

What you do:

- `#include <stdio.h>`
- Declare a `File *` variable (called as a file handler) for the file reference.

CIT 593

C - 31

## File I/O, continued

What you do next:

- Define the `File *` by calling `fopen`, with the correct mode.
- Check if the file is `NULL`. If so, quit, with some error message.
- Depending on what you want:
  - `fprintf(filepointer, "Print this into the file.\n");`
  - `fgets(s, n, filepointer);`
- **ALWAYS** call `fclose(filepointer);` at the end.

CIT 593

C - 32

## textcopy.c

```
#include <stdio.h>

/* Library function prototypes
FILE * fopen(const char * filename, const char *
mode);
int fclose(FILE * stream); */

/* Local function prototypes */

int main(int argc, char * argv[])
{
    char * inputname;
    char * outputname;

    FILE * inputfile;
    FILE * outputfile;

    int charread = 0;
    if (argc != 3)
    {
        printf("Usage: textcopy inputfile outputfile.\n");
        return 1;
    }
    inputname = argv[1];
    outputname = argv[2];

    /* Open/ create files. */
    inputfile = fopen(inputname, "r"); /* mode "Read" */
    outputfile = fopen(outputname, "w"); /* mode "Write" */

    if (inputfile == NULL || outputfile == NULL) /* files did not
open */
    {
        printf("Files could not be opened\n");
        return 1; /* quit now */
    }

    while ((charread = fgetc(inputfile)) != EOF)
    {
        printf("%c", charread);
        fprintf(outputfile, "%c", charread);
    }

    /* close the file */
    if (fclose(inputfile) != 0 || fclose(outputfile) != 0)
    {
        printf("Close file error.");
    }
    return 0;
}

<prompt> ./copy in out
```

Reads & gets a character from file

Contents of file "in" are copied to file "out"

CIT 593

C - 33

## make

- make is a compilation tool.
  - Think of it as the **script** command in PennSim
- It is helpful when you are working on a big project and you have many files to compile & link

### You need:

- Your source code, and/or other required files.
- A text file: makefile
- make executable somewhere in your path.

### You do:

- make

CIT 593

C - 34

## makefile

- makefile is read by make executable to compile, assemble, link and output your program correctly.
  - Think of it as the lcs file in LC3
  - The file you write is called "makefile" without any file extensions
- It can be used to compile programs ranging from helloworld.c to the Linux system itself.
- Write makefile correctly, and all you do is:

### make

CIT 593

C - 35

## makefile structure

```
# This is a line of comment.
# The next line says which compiler to use.
COMPILER=gcc
# Later when you need system libraries, put it here.
# ex. LIBS=-lpthread
LIBS=
# These are gcc options:
# -g produce debugging info useful when using debugger
tools (e.g. gdb)
# -Wall gives all warning.
OPTIONS=-g -Wall
```

CIT 593

C - 36

## makefile, continued

```
# This is the default compilation.
# When it goes beyond 1 line, tells make to continue onto the next line
# by putting \ at the end of the line
all:
[Tab char]$(COMPILER) $(LIBS) $(OPTIONS) \
hello.c -o hello

# This is compiled when clean parameter is given to make,
# e.g. make clean.
clean:
[Tab char] rm -f *.o hello
```

CIT 593

C - 37

## A complete makefile example

```
COMPILER=gcc
LIBS=-lpthread
OPTIONS=-g -Wall

all:
[\\t]$(COMPILER) $(LIBS) $(OPTIONS) sched.c func1.c \
func2.c func3.c func4.c -o sched

clean:
[\\t]rm -f *.o sched
```

**REMEMBER:** makefile is without any extensions

CIT 593

C - 38

## Compiling textcopy.c with make

```
COMPILER=gcc          Perform:
LIBS=                 <prompt> make
OPTIONS=-g -Wall     <prompt> ./textcopy
NAME=textcopy
all:
    $(COMPILER) $(LIBS) $(OPTIONS) $(NAME).c -o
    $(NAME)
op1:
    $(COMPILER) $(LIBS) $(OPTIONS) -S $(NAME).c -o
    $(NAME).s
clean:
    rm -f *.o $(NAME)
```

CIT 593

C - 39

## References

This is the most important section of this tutorial.

Interesting books/readings:

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

Websites:

- Dr.David Matuszek's Concise C: <http://www.cis.upenn.edu/~palsetia/Crefguide.htm>
- The Single UNIX Specification, v2: <http://www.opengroup.org/onlinepubs/007908799/>
- GCC: <http://www.hmug.org/man/1/gcc.php>

CIT 593

C - 40