CSE 220 - C Programming
Fall 2015
Fundamentals
Compiling Source Code

```
#include <stdio.h>
int main(void) {
    printf("Hello World!\n");
    return 0;
}
```

```
gcc HelloWorld.c -o HelloWorld.exe
"-o": specify the name of the output file
```
Compiling Source Code

• From source code to executable

1. Preprocessing
   • Modify or add things to your source code
   • Based on the "directives" you put in your code

2. Compiling
   • Checks if your code followed all the rules of C
   • If no errors => translate source code to machine language ("object code")
   • If errors found => returns list of errors

3. Linking
   • Combines your object code with other code needed
Compiling Source Code

(Image modified from www.aboutdebian.com)
Announcements

» HW #2 will be posted today later

» Project #1 will be posted today later
Fundamentals of C

- Variables
- Statements
- Functions
- Directives
- Comments
First Program Explained

```c
#include <stdio.h>
int main(void) {
    printf(“Hello World!\n”);
    return 0;
}
```

`#include <stdio.h>`: information in the header `stdio.h` needs to be included before the program is compiled

`main`: a function, the main position to start the program at

`: `: delimit start and end of a function

`printf`: a function, displays Hello World to the screen
Fundamentals of C

```c
#include <stdio.h>
int main(void) {
    printf("Hello World!\n");
    return 0;
}
```

- **main**: function, returns a value
- **int**: return value is an integer
- **void**: main does not take any arguments
- **return**: terminates the function, returns a value
- **printf**: a call to function printf
Fundamentals of C

```c
#include <stdio.h>
int main(void) {
    printf("Hello World!\n");
    return 0;
}
```

- **statements:**
  - commands to be executed when the program runs
  - End with semicolon (with some exceptions)
Fundamentals of C

```c
#include <stdio.h>
int main(void) {
    printf("Hello World!\n");
    return 0;
}
```

directives:
- commands for the preprocessor
- one line long
- begin with #
- No semicolon at the end
Fundamentals of C

- Variables
- Statements
- Functions
- Directives
- Comments
/**************
* Compute the square root
* of number 5
* *****************/
#include <stdio.h>
#include <math.h>
int main(void) {
    float x;
    x = sqrt(5);
    printf("The square root of 5 is %f \n", x);
    return 0;
}
/* ******************************
 *  Compute the square root
 *    of number 5
 *  *******************************/

#include <stdio.h>
#include <math.h>

int main(void) {
    float x;
    x = sqrt(5);
    printf("The square root of 5 is \%f \n", x);
    return 0;
}
Fundamentals of C

1: /* ****************************
2: * Compute the square root
3: * of number 5
4: * ****************************/
5: #include <stdio.h>
6: #include <math.h>
7: int main(void) {
8:   float x;
9:   x = sqrt(5);
10:  printf("The square root of 5 is %f \n", x);
11:  return 0;
12: }

Directives
/*  *******************************
 *  Compute the square root
 *    of number 5
 *  *******************************/

#include <stdio.h>
#include <math.h>

int main(void) {
    float x;
    x = sqrt(5);
    printf("The square root of 5 is %f \n", x);
    return 0;
}

Functions
Variables are simply names used to refer to some location in memory - a location that holds a value with which we are working.
A statement is a command given to the computer that instructs the computer to take a specific action, such as display to the screen, or read input from keyboard.
Announcements

» The due date of HW #2 is on September 20
Fundamentals of C

- Variables
- Statements
- Functions
- Directives
- Comments
Variables

• Must have a **type**
  – I.e., what kind of information is stored
    • `int a`: Integer values like 1, 44, 26, 0, 2015, -1, ...

• `float a`: Decimal fractions like 0.1, 2.3, 3.14, 123.456, -0.1, ...
  – "float" is the short name for "floating-point", that means how real number is stored in computers

• `char a`: Characters like a, b, $, #, \n, 'a', 'z', 'A', '$', ...
Variables

• Must have a *type*
  – int, float, char, ...

• Must be *declared* before they can be used
  – You must designate a place before you can store stuff in

Why do we need to declare a variable’s *type*?

for an integer type, compiler usually reserves 4 bytes (compiler dependent)
Sizeof()

`sizeof()`: determines the amount of space a designated datatype would occupy in memory.

```
#include<stdio.h>  /*Header file*/
int main() /* The main function */
{
    int x;       /*Variable Declaration*/
    char a;
    printf("x is %d bytes\n", sizeof(x));
    printf("a is %d bytes\n", sizeof(a));
    return 0;
}
```
Sizeof()

sizeof(): determines the amount of space a designated datatype would occupy in memory.

```c
#include<stdio.h> /*Header file*/
int main() /* The main function */
{
    int x;/*Variable Declaration*/
    char a;
    printf "x is %d bytes\n", sizeof(x));
    printf "a is %d bytes\n", sizeof(a));
    return 0;
}
```

x is 4 bytes
a is 1 bytes
Variables

• **Must have a** *type*  
  – int, float, char, ...

• **Must be** *declared* **before they can be used**  
  – You must designate a place before you can store stuff in

  ```
  float profit;
  int height, width;
  ```

• **Assignment**  
  – Store a value into a variable

  ```
  height = 3;
  profit = 235.2;
  ```
Variables - Case Sensitive

• Hold values, must be declared before use

• Example
  • int a = 3 + 4; /* a is a variable */
  • int A = 10; /* A is a variable */

• what is the value of a?
printf

- **printf** function
- Defined in stdio.h
- Prints the string enclosed in quotation marks
- To print the value of a variable:
  ```c
  int x = 2;
  printf("The value is x");
  ```
  vs.
  ```c
  printf("The value is %d", x);
  ```
  The value is x
  The value is 2
Adding two numbers

\[
\begin{align*}
x & : 5 \\
y & : 10 \\
z & : \text{sum of } x \text{ and } y
\end{align*}
\]

\[
\begin{align*}
\text{int } x, y, z; \\
x &= 5; \\
y &= 10; \\
z &= x + y;
\end{align*}
\]
```c
#include <stdio.h>

int main ()
{
    int x, y, z;
    x=5;
    y=10;
    z=x+y;
    printf ("The sum of %d and %d is %d", x, y, z);
    return 0;
}
```

**Output**

```
The sum of 5 and 10 is 15
```
printf

```
#include <stdio.h>
int main ()
{
    int x,y,z;
    x=5;
    y=10;
    z=x+y;
    printf ("The sum of %d and %d is %d", x, y, z);
    return 0;
}
```

Output

The sum of 5 and 10 is 15
Placeholders

- %d  int
- %f  float
- %c  char
printf

Example

```c
#include <stdio.h>
int main ()
{
    int x,y,z;
    x=5;
    y=10;
    z=x+y;
    printf ("Hello!\n");
    printf ("The sum of %d and %d is %d\n", x,y,z);
    return 0;
}
```

Output
printf

Example

```c
#include <stdio.h>
int main ()
{
    int x, y, z;
    x = 5;
    y = 10;
    z = x + y;
    printf ("Hello!\n");
    printf ("The sum of %d and %d is %d\n", x, y, z);
    return 0;
}
```

Output

Hello! The sum of 5 and 10 is 15
printf

Example

```c
#include <stdio.h>
int main ()
{
    int x, y, z;
    x = 5;
    y = 10;
    z = x + y;
    printf ("Hello!\n");
    printf ("The sum of %d and %d is %d\n", x, y, z);
    return 0;
}
```

Output
printf

Example

```c
#include <stdio.h>
int main ()
{
    int x,y,z;
    x=5;
    y=10;
    z=x+y;
    printf ("Hello!\n");
    printf ("The sum of %d and %d is %d\n", x,y,z);
    return 0;
}
```

Output
Hello!
The sum of 5 and 10 is 15
Variables

• Example: compute volume of a box

```c
#include <stdio.h>
int main ()
{
    int height, width;
    height = 3;
    width = 2.5;
    length = 4;
    int length;
    volume = height * width * length;
    printf("volume is: %d\n", volume);
}
```

• Any errors?
Variables

• Example: compute volume of a box

```c
#include <stdio.h>
int main ()
{
    int height, width;
    height = 3;
    width = 2.5;
    length = 4;
    int length;
    volume = height * width * length;
    printf("volume is: %d\n", volume);
}
```
Variables

• **Initialization**: gives a variable a default value when you declare it

• **Multiple variables** can be initialized in one declaration

• **Uninitialized variable**:  
  – without a default value  
  – Unpredictable result!

If you define variable without initialization and print it on the screen, what would be the output?
Variables

• Uninitialized variable: int x;

```
  20   5001   700  -1256
```

• Initialize the variable: int x = 24;

```
  20   24   700  -1256
```

A random value!
What's wrong here?
Example

Compile time error versus Runtime error

No compiler error
Logical error

<6 arctic:~/volume >gcc volume.c -o volume.exe
<7 arctic:~/volume >ls -l
total 12
-rw------- 1 forsati gnats 183 Sep 9 01:22 volume.c
-rwx------- 1 forsati gnats 6786 Sep 9 01:22 volume.exe
<8 arctic:~/volume >

<9 arctic:~/volume >/volume.exe
Volume: 327660
<10 arctic:~/volume >
Constants

- **Constant**: the same value that will be used many times

- **Macro definition**: use to name constants

```c
#define PI 3.1415

int main(){
    a = PI * radius * radius; /* a = 3.1415 * radius * radius; */
}
```
Constants

• **Constant:** the same value that will be used many times

• **Macro** definition: use to name constants

```c
#define PI 3.1415

int main(){
    a = PI * radius * radius; /* a = 3.1415 * radius * radius; */
    c = 2 * PI * radius; /* c = 2 * 3.1415 * radius*/
}
```
Example

```c
//Constants
#include<stdio.h>
#define PI 3.14  /* PI is a constant */

int main ()
{
    int r =2;
    float area = PI * r * r;
    printf ("Area = %f\n", area);
}
```

Output

Area = 12.560000
Example

```c
//Constants
#include<stdio.h>
#define PI 3.14 /* PI is a constant */

int main ()
{
    int r =2;
    float area = PI * r * r;
    printf ("Area = %d\n", area);
}
```

What's wrong here?
Example

```c
//Constants
#include<stdio.h>
#define PI 3.14 /* PI is a constant */

int main ()
{
    int r = 2;
    float area = PI * r * r;
    printf("Area = %d\n", area);
}
```

Area = 1508432888!!!!!!!
Names for Constants

• The preprocessor replaces every occurrence by the value it represents

• Convention: use all capital letters for constant names

• If expression contains operators it should be enclosed by parentheses
  
  #define SCALE_FACTOR (5.0 / 9.0)
Reading Input

• `scanf` function: reads the value entered by the user
  • `scanf("%d", &x)`:  
    – reads an integer and stores it in variable `x`
  • `scanf("%f", &y)`:  
    – reads a float and stores it in variable `y`

• Note that variables should be declared first:  
  ```
  int x;
  float y;
  ```
```c
#include<stdio.h> /*Header file*/
int main() /* The main function */
{
    int x, y, z; /*Variable Declaration*/
    printf("Enter x:");
    scanf("%d", &x); /* Wait for input */
    printf ("Enter y:");
    scanf ("%d", &y); /* Wait for input */
    z = x + y;
    printf ("The sum is %d", z);
}
```
```c
#include<stdio.h> /*Header file*/

int main() /* The main function */
{
    int x, y, z; /*Variable Declaration*/
    printf("Enter x:");
    scanf("%d", &x); /* Wait for input */
    printf("Enter y:");
    scanf("%d", &y); /* Wait for input */
    z = x + y;
    printf("The sum is %d", z);
}
```

Output

Enter x:12
Enter y:23
The sum is 35
```c
#include <stdio.h>

int main()
{
    int volume, height, length, width;

    printf("Enter the height: \n");
    scanf("%d", &height);
    printf("Enter the length: \n");
    scanf("%d", &length);
    printf("Enter the width: \n");
    scanf("%d", &width);

    volume = height * length * width;
    printf("Volume: %d\n", volume);

    return 0;
}
```
Output 1

Enter the height: 12
Enter the length: 18
Enter the width: 14
Volume: 3024

Output 2

Enter the height: -1
Enter the length: 12
Enter the width: 4
Volume: -48
Fundamentals of C

- Variables
- **Statements**
- Functions
- Directives
- Comments
A **statement** is a command given to the computer that instructs the computer to take a specific action, such as display to the screen, or read input from keyboard.
Statements

• The "actions" you perform using your code
  – Declare a variable
  – Assign a value
  – Do some calculation
  – Call a function
  – etc.

• Remember to end a statement with semicolon ;
Fundamentals of C

- Variables
- Statements
- **Functions**
- Directives
- Comments
Functions

• A function is a "procedure" or "sub-routine"
  – A sequence of statements, that accomplishes a specific "task"
  – Can take inputs and give an output

```c
7:    int main(void) {
8:       float x;
9:       x = sqrt(5);
10:     printf("The square root of 5 is %f \n", x);
11:     return 0;
12:   }
```
Functions

• The way to logically organize your code
  – Easier to understand
  – Easier to re-use

• C is a "function-based" language
  – All you do is to write functions, and put them together
Functions

• The "main" function
  – Required for a C program
  – The "entry point"

... ...
7:   int main(void) {
8:     float x;
9:     x = sqrt(5);
10:    printf("The square root of 5 is %f \n", x);
11:    return 0;
12: }

{ }: delimit start and end of a function
Fundamentals of C

• Variables
• Statements
• Functions
• Directives
• Comments
Directives

• Directives
  – Commands for the preprocessor
    • The preprocessor will "edit" your source code based on the directives
  – One line long
  – Begin with #
  – NO semicolon at the end
Directives

1: /* ******************************************
2: * Compute the square root
3: * of number 5
4: * ******************************************/
5: #include <stdio.h>
6: #include <math.h>
7: int main(void) {
8:     float x;
9:     x = sqrt(5);
10:    printf("The square root of 5 is %f \n", x);
11:    return 0;
12: }

stdio.h
math.h
Fundamentals of C

- Variables
- Statements
- Functions
- Directives
- Comments
Comments

• Provide documentation
• Ignored by the compiler
• May appear anywhere
• May extend over multiple lines (/**/)
  – Cannot be nested
• // comments end at the end of the line

/* HelloWorld.c
   Purpose: prints greeting */
#include <stdio.h>
int main(void) {
   /* Greet twice */
   printf("Hello World!\n"); // 1st
   printf("Hello World!\n"); // 2nd
   return 0;
}
Comments

- Where does the first comment end?
- Where does the third comment end?
- What's wrong?
  - Cannot be nested
- Code editors (e.g., gedit) use different colors for comments. Help tracking comment termination.
Comments

```c
/* HelloWorld.c
   Purpose: prints greeting */
#include <stdio.h>
int main(void) {
   /* Greet twice */
   printf("Hello World!\n"); // 1st
   printf("Hello World!\n"); // 2nd
   return 0;
}
```

- Provide documentation
- Ignored by the compiler
- May appear anywhere
- May extend over multiple lines (/***/)
  - Cannot be nested
- // comments end at the end of the line
Comments

1: /* **************
2: * HelloWorld.c *
3: * Purpose: prints greeting *
4: * *********************/
5: #include <stdio.h>
6: int main(void) {
7:  /* Greeting #1 */
8:  printf("Hello World!\n");
9:  /* Do not show 2\textsuperscript{nd} greeting */
10:  /* Greeting #2 */
11:  printf("Hello World!\n"); */
12:  return 0;
13: }

• Where does the first comment end?

• Where does the third comment end?

• Code editors (e.g., gedit) use different colors for comments. Help tracking comment termination.
Fundamentals of C

• Variables
• Statements
• Functions
• Directives
• Comments

Now you know everything about C ^_^
Identifiers

• Names for macros, variables, functions
• May contain letters, digits and underscores
• Must begin by letter or underscore
• Case sensitive
• Valid names:
  – x, y, first_name, lastName, _age, value1, value2, steelboxwidth, steelBoxWidth
• Invalid names:
  – 1st_value, last-name, scale factor
• Conventions:
  – lower case, separate by underscore: box_height, box_width
  – lower case, separate by uppercase: boxHeight, boxWidth
Keywords

- Special words in C
- Cannot be used as identifiers

<table>
<thead>
<tr>
<th>auto</th>
<th>break</th>
<th>case</th>
<th>char</th>
<th>const</th>
</tr>
</thead>
<tbody>
<tr>
<td>continue</td>
<td>default</td>
<td>do</td>
<td>double</td>
<td>else</td>
</tr>
<tr>
<td>enum</td>
<td>extern</td>
<td>float</td>
<td>for</td>
<td>goto</td>
</tr>
<tr>
<td>if</td>
<td>int</td>
<td>long</td>
<td>register</td>
<td>return</td>
</tr>
<tr>
<td>short</td>
<td>signed</td>
<td>sizeof</td>
<td>static</td>
<td>struct</td>
</tr>
<tr>
<td>switch</td>
<td>typedef</td>
<td>union</td>
<td>unsigned</td>
<td>void</td>
</tr>
<tr>
<td>volatile</td>
<td>while</td>
<td>restrict</td>
<td>_Bool</td>
<td>_Complex</td>
</tr>
<tr>
<td>inline</td>
<td>restrict</td>
<td>_Bool</td>
<td>_Complex</td>
<td>_Imaginary</td>
</tr>
</tbody>
</table>
Exercise

• Which of the following are not keywords in C?
  a) For
  b) for
  c) printf
  d) while
  e) int
Layout of a C program

• Readability is important
• Indent your program
• Add blank lines
• Divide long statements into multiple lines
  
  printf("The volume of the box in cubic feet is %d",
  height*width*length);

• Cannot add spaces in the middle of a token
  
  printf("The volume of the box in cubic feet is %d", height*width*length); /* wrong */
Bad Example

```c
int main(void) {
    int a, h, b;
    printf("Enter the height:\n");
    scanf("%d", &h);
    printf("Enter the length:\n");
    scanf("%d", &b);
    a = h * b / 2;
    printf("Area: %d\n", a);
    return 0;
}
```
int main(void) {
    int volume, height, length, width;

    /* Read values from user */
    printf("Enter the height:\n");
    scanf("%d", &height);
    printf("Enter the length:\n");
    scanf("%d", &length);
    printf("Enter the width:\n");
    scanf("%d", &width);

    /* Compute the volume and print it */
    volume = height * length * width;
    printf("Volume: %d\n", volume);

    return 0;
}