C Programming Basics

Ritu Arora

Texas Advanced Computing Center

June 5th, 2012

Email: rauta@tacc.utexas.edu

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

Overview of the Course

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts will be accompanied with examples.

Creating a C Program

- Have an idea about what to program
- Write the source code using an editor or an Integrated Development Environment (IDE)
- Compile the source code and link the program using a C compiler
- Fix errors, if any
- Run the program and test it
- Fix bugs, if any

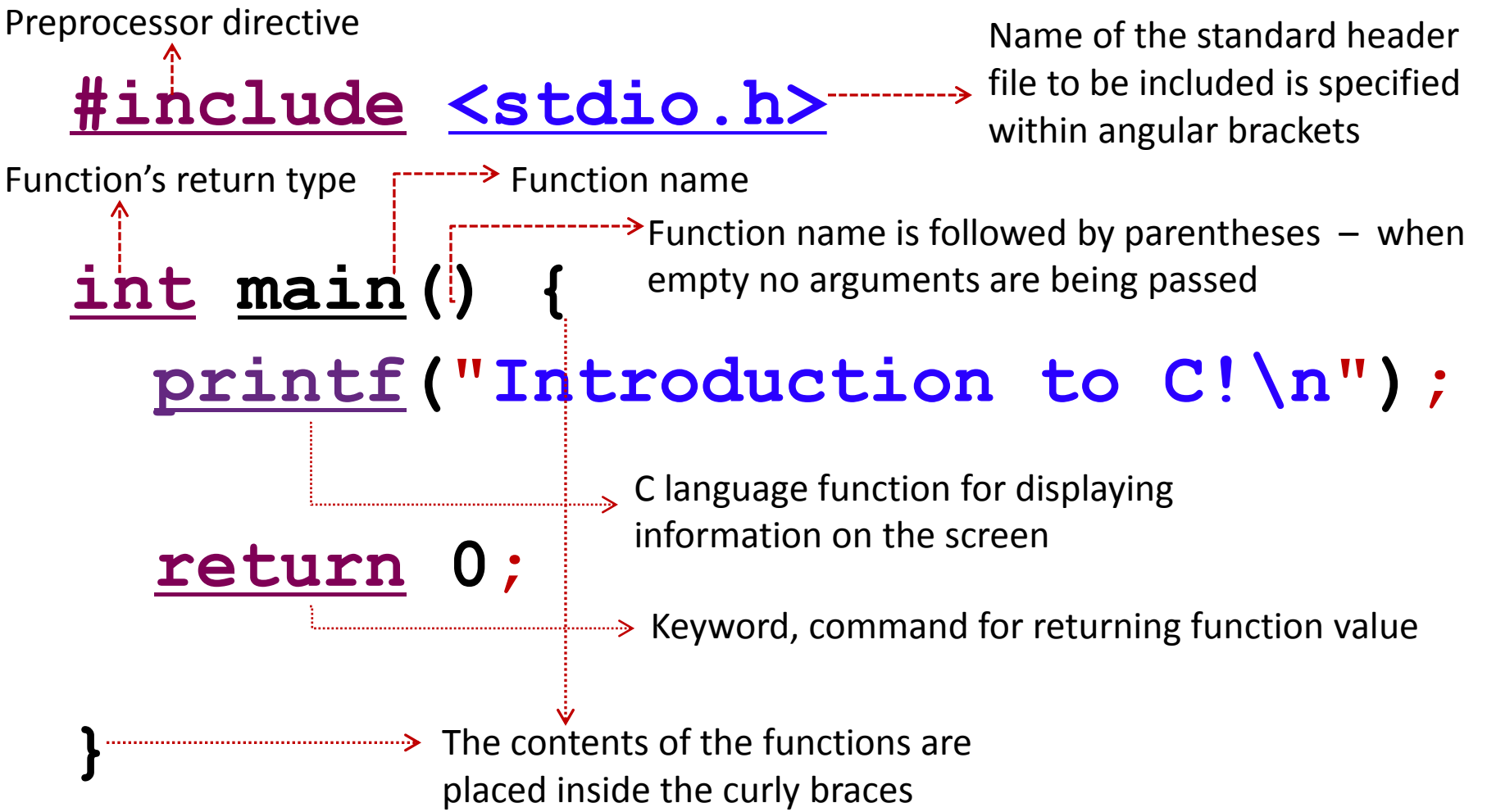
Write the Source Code: firstCode.c

```
#include <stdio.h>

int main(){

   printf("Introduction to C!\n");

   return 0;

}
```

Output:

```
Introduction to C!
```

Understanding firstCode.c



Text strings are specified within `""` and every statement is terminated by `;`
Newline character is specified by `\n`

Save-Compile-Link-Run

- Save your program (source code) in a file having a “c” extension.

  Example, `firstCode.c`

- Compile and Link your code (by default, GCC automatically does the linking)

```
gcc -o firstCode firstCode.c
```

- Run the program

```
./firstCode
```

Repeat the steps above every time you fix an error!

Different Compilers

- Different commands for different compilers (*e.g.*, **`icc`** for intel compiler and **`pgcc`** for pgi compiler)
  - GNU C program

```
gcc -o firstCode firstCode.c
```

  - Intel C program

```
icc -o firstCode firstCode.c
```

  - PGI C program

```
pgcc -o firstCode firstCode.c
```

- To see a list of compiler options, their syntax, and a terse explanation, execute the compiler command with the -help or --help option

Summary of C Language Components

- Keywords and rules to use the keywords
- Standard header files containing functions like **`printf`**
- Preprocessor directives for including the (standard) header files
- Function **`main`**
- Parentheses and braces for grouping together statements and parts of programs
- Punctuation like `;`
- Operators like `+`
- All the above and more to come make up the syntax of C

Pop Quiz
(add the missing components)

```
_________ <stdio.h>

int main()__

  printf("Introduction to C!\n") __

  printf("This is a great class!\n");

  return 0;


__
```

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

Warnings, Errors and Bugs

- Compile-time warnings
  - Diagnostic messages
- Compile-time errors
  - Typographical errors: pirntf , $include
- Link-time errors
  - Missing modules or library files
- Run-time errors
  - Null pointer assignment
- Bugs
  - Unintentional functionality

Find the Error: error.c

```
#include <stdio.h>
int main(){
   printf("Find the error!\n")
   retrun(0);
}
```

Error Message (compile-time error)

```
gcc –o error.o error.c
..\error.c: In function 'main':
..\error.c:4:3: error: expected ';' before 'retrun'
..\error.c:5:1: warning: control reaches end of non-void function
Build error occurred, build is stopped
Time consumed: 148  ms.
```

Find the Error: error.c

```
#include <stdio.h>
int main(){
   printf("Find the error!\n");
   retrun 0;
}
```

Error Message (link-time error)

```
gcc -o error error.c
…
..\error.c:4:3: warning: implicit declaration of function 'retrun'
…
gcc -oCTraining.exe error.o
error.o: In function `main':
C:\Users\ra25572\workspace\CTraining\Debug/../error.c:4: undefined reference to `retrun'
collect2: ld returned 1 exit status
Build error occurred, build is stopped
Time consumed: 436  ms.
```

Find the Error: error2.c

```
#include < stdio.h >
int main(){
   printf("Find the error!\n");
   return 0;
}
```

Error Message (compile-time error)

```
gcc -o error2 error2.c
..\error2.c:1:21: fatal error:  stdio.h : No such file or directory
compilation terminated.
Build error occurred, build is stopped
Time consumed: 98  ms.
```

Overview of the Lecture

- Writing a Basic C Program
- Understanding Errors
- Comments, Keywords, Identifiers, Variables
- Standard Input and Output
- Operators
- Control Structures
- Functions in C
- Arrays, Structures
- Pointers
- Working with Files

All the concepts are accompanied by examples.

Homework 1

- Find the error in the following code

```
#include <stdio.h>
int main(){
 printf(Find the error!\n");
 return(0);
}
```

- Write a program to print your name on the screen (Edit the code on slide # 4).
  - `printf("My name is Ritu.");`

Comments and New Line: rules.c

```
/*
 * rules.c
 * this is a multi-line comment
 */
#include <stdio.h>
int main(){
  printf("Braces come in pairs.");
  printf("Comment tokens come in pairs.");
  printf("All statements end with semicolon.");
  printf("Every program has a main function.");
  printf("C is done mostly in lower-case.");
  return 0;
}
```

Output of rules.c

```
Braces come in pairs.Comment tokens come in
pairs.All statements end with a
semicolon.Every program must have a main
function.C is done mostly in lower-case.
```

Output looks odd! We want to see a new line of text for every `printf` statement.

Comments and New Line: rules.c

```
/*
 * rules.c
 * this is a multi-line comment
*/
#include <stdio.h>
int main(){
 /* notice the \n in the print statements */
 printf("Braces come in pairs.\n");
 printf("Comment tokens come in pairs.\n");
 printf("All statements end with semicolon.\n");
 printf("Every program has a main function.\n");
 printf("C is done mostly in lower-case.\n");
 return 0;
}
// this is another way to specify single-line comments
```

Output of rules.c

```
Braces come in pairs.
Comment tokens come in pairs.
All statements end with a semicolon.
Every program must have a main function.
C is done mostly in lower-case.
```

The output looks better now!

Do-It-Yourself Activity

- Learn the various ways in which you can print and format values of various data types.

- For example:
  - How would you print an integer?
  - How would you print a value of type double with precision of 8 places after the decimal?

- Reference:
  - http://www.cplusplus.com/reference/clibrary/cstdio/printf/

Some C Language Keywords

Category	Keywords
Storage class specifiers	`auto register static extern typedef`
Structure & union specifiers	`struct union`
Enumerations	`enum`
Type-Specifiers	`char double float int long short signed unsigned void`
Type-Qualifiers	`const volatile`
Control structures	`if else do while for break continue switch case default return goto`
Operator	`sizeof`
Deprecated keywords	`fortran entry`
Other reserved words	`asm bool friend inline`

Variables

- Information-storage places
- Compiler makes room for them in the computer's memory
- Can contain string, characters, numbers *etc.*
- Their values can change during program execution
- All variables must be declared before they are used and must have a data type associated with them
- Variable must be initialized before they are used

Data Types

- Data types specify the type of data that a variable holds

- Categories of data types are:
  - **Built-in:** `char double float void int (short long signed unsigned)`
  - **User-defined:** `struct union enum`
  - **Derived:** `array function pointer`

- We have already seen an example code in which an integer data type was used to return a value from a function:

```
int main()
```

- Compiler-dependent range of values associated with each type. For example: an `int` can have a value in the range
  - **–32768** to **32767** on a 16-bit computer or
  - **–2147483647** to **2147483647** on a 32-bit computer

Identifiers

- Each variable needs an identifier (or a name) that distinguishes it from other variables
- A valid identifier is a sequence of one or more letters, digits or underscore characters
  - Note: you cannot begin with a digit
- Keywords cannot be used as identifiers

Variable Declaration

- Declaration is a statement that defines a variable
- Variable declaration includes the specification of data type and an identifier. Example:

```
int number1;
float number2;
```

- Multiple variables can be declared in the same statement

```
int x, y, z;
```

- Some types of data can be signed or unsigned
- Signed types can represent both positive and negative values, whereas unsigned types can only represent positive values

```
signed double temperature;
```

Variable Initialization

- A variable can be assigned a value when declared
  - Assignment operator is used for this purpose
  - `int x = 10;`

- More examples
  - `char x = 'a';`
  - `double x = 22250738585072014.e23;`
  - `float x = 10.11;`

- `void` cannot be used to declare a regular variable
  - It is used as a return type of a function or as an argument of a function

Example of Updating Variables: myAge.c

```
#include <stdio.h>
int main(){
   int age;
   age = 10;
   printf("Initial value of age is: %d\n", age);
   age = 20;
   printf("Updated value of age is: %d\n", age);
   age = age + 20;
   printf("New updated value of age is: %d\n", age);
   return 0;
}
```

Output:

```
Initial value of age is: 10
Updated value of age is: 20
New updated value of age is: 40
```

Scope of Variables

- A variable can be either of global or local scope
  - Global variables are defined outside all functions and they can be accessed and used by all functions in a program file
  - A local variable can be accessed only by the function in which it is created
- A local variable can be further qualified as **`static`**, in which case, it remains in existence rather than coming and going each time a function is called
  - **`static int x = 0;`**
- A **`register`** type of variable is placed in the machine registers for faster access – compilers can ignore this advice
  - **`register int x;`**

Constants and Constant Expressions

- The value of a constant never changes
  - `const double e = 2.71828182;`

- Macros
  - `#define MAXRECORDS 100`
  - In the code, identifiers (`MAXRECORDS`) are replaced with the values (`100`)
  - Helps to avoid hard-coding of values at multiple places
  - Example: `char records[MAXRECORDS + 1];`
  - Can be used at any place where constants can be used

- Enumeration is a list of constant values
  - `enum boolean {NO , YES};`

Expressions containing constants are evaluated at compile-time

References

- C Programming Language, Brian Kernighan and Dennis Ritchie
- Let Us C, Yashavant Kanetkar
- C for Dummies, Dan Gookin
- http://cplusplus.com