A UT/ORNL PARTNERSHIP
NATIONAL INSTITUTE FOR COMPUTATIONAL SCIENCES
NICS

Intro to C Programming

Ryan Hulguin

What is C

- A general-purpose programming language initially developed by Dennis Ritchie at AT&T Bell Labs
- Designed to produce portable code while maintaining performance and minimizing footprint
- Requires compiler to generate executable
- Provides low level memory access
- Many operating systems are written in C
- Perl, PHP, and Python are also written in C

C Programming Tools

Required

- **Text Editor - vi**
- **Compiler - gcc**

Optional

- **Debugger**
- **Profiler**
- **Integrated Development Environment (IDE)**

C Program Structure

These are the basic parts of a program:

- Preprocessor Commands
- Functions
- Variables
- Statements and Expressions
- Comments

Program Example 1

```
#include <stdio.h>
// Program execution starts at the main function
int main()
{
   int programNumber = 1;
   printf("This is program number %d.\n", programNumber);

   return 0;
}
```

Using a text editor create the file example1.c with the code above

Compiling and running Example 1

- vi example1.c
- gcc –o example1.exe example1.c
- ./example1.exe

Breaking down Program example 1

- The first line is a preprocessor command that tells the compiler to include the stdio.h file before compiling
  - This enables use of the printf() function
- The second line is a comment and is ignored by the compiler. Comments are meant to improve human readability
- The third line is the main function. Execution starts there
- The curly brackets { } group together related statements, like several sentences make up a paragraph

Breaking down Program example 1

- An integer variable is defined and initialized to a value of 1
- The function printf() is called with 2 arguments
- It writes a message to stdout, which by default is the console display
- The main function returns a value of 0 and the program terminates

Tokens

- Tokens are the basic building blocks of C programs
- There are 6 types of C tokens:

1. keyword – reserved words in C
2. identifier – names of functions/variables
3. constant – hard coded numbers, i.e. 1
4. string literal – words or sentences in quotes
5. symbol/separator – example ( ) { } ,
6. operator – example + - * /

Token Breakdown of printf line

- The following code contains 7 tokens

```
printf("This is program number %d.\n",
programNumber);
```

1. printf
2. (
3. "This is program number %d. \n"
4. ,
5. programNumber
6.)
7. ;

Identifiers

- 2 of the tokens are identifiers

`printf` and `programNumber`

- Identifiers identify a variable, function, or any other user defined item
- Identifiers can only start with
  - letters A to Z
  - letters a to z
  - underscore
- C is case sensitive

Reserved Keywords

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Statements and comments

- Each individual statement ends with a semicolon
- More than 1 statement can be on a line

```
double a=1.0; int b=4;
```

- Comments are ignored by compiler
- Single comments begin with //
- Multiple line comments can be blocked off with /* and */

Comments

```
// This is a comment
/* This is also a comment */
/* This is one big comment block
double a=4.0;
The above line is ignored since it
resides in the comment block
This ends the comment block */
```

Whitespace

- Whitespace separates one part of a statement from another
- space, tab, and newline/EOL characters are all whitespace
- whitespace helps/hurts with code readability
- The following are equivalent legal statements:

```
int a = 3;

int a=3;

int a

=

3;
```

C Data Types

- Basic Types
  - integer types and floating-point types
- Enumerated types
  - used to define variables that can only have certain discrete integer values
- void type
- Derived types
  - advanced data types including pointers, arrays, structures, unions, and functions

Integer Types

Type	Range of values
char	-128 to 127 or 0 to 255
unsigned char	0 to 255
signed char	-128 to 127
int	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	0 to 65,535
short	-32,768 to 32,767
unsigned short	0 to 65,535
long	-2,147,483,648 to 2,147,483,647
unsigned long	0 to 4,294,967,295

Floating-Point Types

Type	Range of values	Precision
float	1.2E-38 to 3.4E+38	6 decimal places
double	2.3E-308 to 1.7E+308	15 decimal places
long double	3.4E-4932 to 1.1E+4932	19 decimal places

C Variables

- Variables are storage areas that hold values
- All variables must be declared before use

```
type variable_list;
int a, b, c;
float d, e, f;
```

- Variables can be intialized upon declaration or afterwards

```
int a = 23;
int b;
// Do stuff
b = 45;
```

Typecasting

- Sometimes you want to change one basic type to another

`(int) 3.75` evaluates to simply `3`

`(double) 5` evaluates to `5.0`

Arithmetic operators

A *op* B

- **+**
- **-**
- *****
- **/**
- **%**

Increment and decrement operators

- **A++**
- **B--**

Relational Operators

(A *op* B)

- ==
- !=
- >
- <
- >=
- <=

Evaluates to 1 if true, 0 if false

Logical Operators

```
(A && B) // true if both true
(A || B) // true if at least 1 is true
!(  ) // returns the opposite
```

Assignment operators

```
=     A = B + C

+=    B += A    is the same as    B = B + A

-=    B -= A    is the same as    B = B - A

*=    B *= A    is the same as    B = B * A
```

if statement

```
if (boolean_expression)
{
      // Do stuff if expression is true
      // Do even more stuff
}
//If only one statement is to be executed,
leave out { }
if (A < B)
      A = A + 1;
B = B-1; // This statement executed always
```

if else

```
if (boolean_expression)
{
      // Do stuff if expression is true
      // Do even more stuff
}
else
{
      // Do this instead if expression is false
}
```

if else if

```
if( this_is_true )
{
     do this; and that; and this;
}
else if( this_is_true_instead )
{    do this; and that; and this;  }
else
{    } // Nothing else matters in this example
```

? : operator

```
expression1 ? expression2 : expression3;
// This is the same as:
if (expression1)
{
     expression2;
}
else
{
     expression3;
}
```

while loop

```
while (expression)
{
      // Do stuff while the expression is true
      // Ideally you will want to provide a way
      // For the expression to evaluate to false
      /* So you don't have an endless loop */
}
```

do … while loop

```
do
{
      statement1;
      statement2;
      statement3;
} while( expression );


// This will execute all 3 statements once
before the expression is tested to be true
```

for loop

```
for ( init; condition; increment )
{
     // Statements to execute
}
```

```
int n;
for ( n = 0; n < 10; n++ ) // Loop 10 times
{
     printf("n = %d.\n", n);
}
```

break and continue

- `break` lets you immediately terminate the loop and execute the code after the loop
- `continue` ignores all remaining statements in a loop and then returns to the top of the loop

Functions

```
return_type function_name ( parameter list)
{
      // Body of the function
}
```

String Formatting

code	type	format
d	int	decimal (base ten) number
o	int	octal number (no leading '0' supplied in printf)
ld	long	decimal number
u	unsigned	decimal number
lu	unsigned long	decimal number
c	char	single character
s	char pointer	string
f	float	number with six digits of precision
g	float	number with up to six digits of precision
e	float	number with up to six digits of precision, scientific notation
lf	double	number with six digits of precision
lg	double	number with up to six digits of precision
le	double	number with up to six digits of precision, scientific notation

- Earlier we used %d inside a string literal
- We could also use %lf when trying to output a double

```
double myPi = 3.14;
printf("pi = %lf\n", myPi);
```

Static Arrays

Static arrays are created with the bracket notation

int A[10]; // Creates an integer array with 10 entries

A[0] is the 1st entry

A[9] is the last entry

Our static array named A can be initialized from 1 to 10 with

```
int n;

for ( n = 0; n < 10; ++n) { A[n] = n + 1; }
```

Structures

- structures allow you to combine data items of different types

```
struct [structure tag]
{
    member definition;
    member definition;
    ...
    member definition;
} [optional structure variables];
```

Employee Struct

```
struct Employee
{
    int age;
    double weight;
} Bob, Jim;
// Create a new employee in addition to Bob
// and Jim
struct Employee Richard;
```

Using members of Employee structs

```
Bob.age=27;

Bob.weight=202.3;

Jim.age=43;

Jim.weight=167.4;

Richard.age=18;

Richard.weight=337.1;


printf("Richard is %d years old and weighs %lf
pounds.\n",Richard.age, Richard.weight);
```

serialflops example

- The next few slides is the source code for a C program that takes an array, scales it by 1.1, and then adds another array
- In the main loop 2 floating operations are performed (multiply and add)
- Using built in time functions, the number of gigaflops per second are computed
- Note that is a serial example, and only runs on 1 core of a multicore CPU.
- Compile it with aggressive optimizations and note the performance increase

```
gcc -O3 serialflops.c
```

serialflops.c example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

// Computes current wall clock time
double dtime()
{
   double tseconds = 0.0;
   struct timeval mytime;
   gettimeofday(&mytime,(struct timezone*)0);
   tseconds = (double)(mytime.tv_sec + mytime.tv_usec * 1.0e-6);
   return(tseconds);
}
```

```
// Preprocessor definitions
#define FLOPS_ARRAY_SIZE (1024*1024)
#define MAXFLOPS_ITERS 1000000
#define LOOP_COUNT 128
// Floating point operations per second
#define FLOPSPERCALC 2

// Define static arrays
float fa[FLOPS_ARRAY_SIZE]; //float fa[(1024*1024)]
float fb[FLOPS_ARRAY_SIZE];

int main()
{
   int i,j,k;
   double tstart, tstop, ttime;
   double gflops=0.0;

   float a = 1.1;
```

```
// Initialize the arrays
  for (i=0; i<FLOPS_ARRAY_SIZE; ++i)
  {
            fa[i] = (float)i + 0.1;
            fb[i] = (float)i + 0.2;
  }

  // Get starting time
  tstart = dtime();

  // Calculate many times
  for (j=0; j<MAXFLOPS_ITERS; j++)
  {
     // Scale the 1st array and add in the 2nd array
     for (k=0; k<LOOP_COUNT; k++)
     {
        fa[k] = a * fa[k] + fb[k]; // 2 floating operations, multiply and add
     }
  }
```

```
 // Get stop time
   tstop = dtime();

   // Calcuate gigaflops
   gflops = (double) (1.0e-9 * LOOP_COUNT * MAXFLOPS_ITERS * FLOPSPERCALC);

   // Total elpased time
   ttime = tstop - tstart;

   // Output GFlops

   if (ttime > 0.0)
   {
      printf("GFlops = %10.3lf, Secs = %10.3lf, GFlops per sec = %10.3lf\r\n", gflops,
ttime, gflops/ttime);
   }

   return (0);
}
```