C Programming Basics

SDS 322/329

October 22, 2015

Email any questions to:
rauta@tacc.utexas.edu

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN

Administrative Trivia

- Quiz#3
  - 78% of the class: >=90%
  - 14% of the class: 80-85%
  - 2% of the class: 75%
  - 2% of the class: 60%

Recap

- In the previous lecture
  - Typecasting
  - Scope of variables
  - Constants
  - Operators
  - Standard I/O functions
    - `scanf`, `fflush`, `gets`, `puts`, `getchar`, `putchar`, `strcpy`

Overview of the Lecture

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

All the concepts are accompanied by examples.

Numbers Entered From Keyboard

- Keyboard input is read as a string
- The integer 25 is different from text “25” entered via keyboard
- Convert string to integer by using the **`atoi`** function
  - It is defined in the header file **`stdlib.h`**
  - The string to be converted by this function should begin with a number
- For other conversion functions see:

http://en.wikibooks.org/wiki/C_Programming/C_Reference/stdlib.h

String to Integer Conversion: strToInt.c

```
#include <stdio.h>
#include <stdlib.h>
int main(){
 int age;
 char enterAge[8];
 printf("How old is your friend?\n");
 fflush(stdout);
 gets(enterAge); // enter the value for age
 age = atoi(enterAge);
 printf("Your friend's age is: %d", age);
 return 0;
}
```

Note: string to integer conversion

Note: string to integer conversion

Output:

```
How old is your friend?
22
Your friend's age is: 22
```

Overview of the Lecture

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

All the concepts are accompanied by examples.

Control Structures

- **Sequence Structure** is a sequence of statements
  - All the programs that we have seen so far in class executed the statements one-by-one until the defined end
  - This type of program is called sequential structure
  - The program flows in single direction
- **Selection Structure** used for branching
- **Loop Structure** used for iteration or repetition

Note: These control structures (just a terminology) should not be confused with the data storage units known as structures in C - those are defined using the `struct` keyword and are used for storing heterogeneous types of values. More on this later.

Selection Structure

- Give the ability to change the flow of program execution
  - achieved by establishing the truth or untruth of an expression (condition)
- This structure allows the program to decide an action based upon user's input or other calculated values
- Types of selection structures that can be implemented in C
  - if selection statement
  - If-else selection statement
  - switch selection statement
- Before we go further into the selection structure, you need to know the meaning of True and False in C programming
  - An expression that evaluates to non-zero (1) is considered true expression while false expression evaluates to zero (0)

Conditional Expressions

- Use **`if-else`** or ternary operator (**`?:`**)

```
if (a > b){
  z = a;
}
```

```
if (a > b){
  z = a;
} else {
  z = b;
}
```

```
z = (a > b) ? a : b ; //z = max (a, b)
```

if-else: Logical Expressions

```
if(temp > 75 && temp < 80){
 printf("It's nice weather outside\n");
}
```

```
if (value == 'e' || value == 'n' ){
  printf("\n Exiting the program.\n");
} else {
  printf("\n In the program.\n");
}
```

Decision Making, Multi-Way Decisions

- Decisions are expressed by **`if-else`** where the **`else`** part is optional

```
if (expression)
   statement1
else
   statement2
```

- Multi-way decisions are expressed using **`else-if`** statements

```
if (expression1)
   statement1
else if (expression2)
   statement2
else
   statement3
```

Multi-Way Decision

- The **`switch`** statement is a multi-way decision
- It tests whether an expression matches one of a number of constant integer values, and branches accordingly

```
switch (expression){
  case const-expression1: statements1
  case const-expression2: statements2
  default: statements3
}
```

Multi-Way Decision Example 1: multiWay1.c

```
char c;
//other code
c = getchar();
if(c=='1')
    printf("Beverage\nThat will be $8.00\n");
else if(c=='2')
    printf("Candy\nThat will be $5.50\n");
else if(c=='3')
    printf("Hot dog\nThat will be $10.00\n");
else if(c=='4')
    printf("Popcorn\nThat will be $7.50\n");
else{
    printf("That is not a proper selection.\n");
    printf("I'll assume you're just not hungry.\n");
    printf("Can I help whoever's next?\n");
}
```

the character read from the keyboard is stored in variable c

If multiple statements depend upon a condition, use { }

//This is just a code snippet. For complete program, see file multiWay1.c

Output of multiWay1.c

```
Please make your treat selection:
1 - Beverage.
2 - Candy.
3 - Hot dog.
4 - Popcorn.
3 <enter>
Your choice:Hot dog
That will be $10.00
```

Multi-Way Decision Example 2: multiWay2.c

```
c = getchar();
switch(c){
  case '1':
    printf("Beverage\nThat will be $8.00\n");
    break;
  case '2':
    printf("Candy\nThat will be $5.50\n");
    break;
  case '3':
    printf("Hot dog\nThat will be $10.00\n");
    break;
  case '4':
    printf("Popcorn\nThat will be $7.50\n");
    break;
  default:
    printf("That is not a proper selection.\n");
    printf("I'll assume you're just not hungry.\n");
    printf("Can I help whoever's next?\n");
}
```

//This is just a code snippet. For complete program, see file multiWay2.c

Loops

- For repeating a sequence of steps/statements
- The statements in a loop are executed a specific number of times, or until a certain condition is met
- Three types of loops
  - `for`
  - `while`
  - `do-while`

`for` Loop

```
for (start_value; end_condition; stride)
    statement;

for (start_value; end_condition; stride) {
    statement1;
    statement2;
    statement3;
}
```

`for` Loop Example 1: forLoop.c

```
#include <stdio.h>
int main(){
  int i;
  for(i = 0 ; i <= 10 ; i = i+2){
    printf("What a wonderful class!\n");
  }
  return 0;
}
```

Output:

```
What a wonderful class!
What a wonderful class!
What a wonderful class!
What a wonderful class!
What a wonderful class!
What a wonderful class!
```

`for` Loop Example 2

```
#include <stdio.h>
int main(){
   int i, sum;
   sum = 0;
   for(i = 1 ; i <= 100 ; i = i+1){
      sum = sum + i;
   }
   printf("Sum of first 100 numbers is: %d ", sum);
   return 0;
}
```

Output:

```
Sum of first 100 numbers is: 5050
```

Did you notice how multiple variables can be declared in the same line?

`while` Loop

- The while loop can be used if you don't know how many times a loop should run

```
while (condition_is_true){
   statement (s);
}
```

- The statements in the loop are executed until the loop condition is true

- The condition that controls the loop can be modified inside the loop (this is true in the case of `for` loops too!)

`while` Loop Example: whileLoop.c

```
#include <stdio.h>
int main(){
  int counter, value;
  value = 5;
  counter = 0;
  while ( counter < value ){
    counter++;  <-- Equivalent to counter = counter +1;
    printf("counter value is: %d\n", counter);
  }
  return 0;
}
```

Output:

```
counter value is: 1
counter value is: 2
counter value is: 3
counter value is: 4
counter value is: 5
```

do-while Loop

- This loop is guaranteed to execute at least once

```
do{
 statement (s);
}
while(condition_is_true);
```

do-while Example: doWhile.c

```
#include <stdio.h>
int main(){
  int counter, value;
  value = 5;
  counter = 0;
  do {
    counter++;
    printf("counter value is: %d\n", counter);
  } while ( counter < value);
  return 0;
}
```

Note the semi-colon after specifying while

Output same as that of the while loop program shown earlier

Keyword: `break`

- **`break`** is the keyword used to stop the loop in which it is present

```
for(i = 10; i > 0; i = i-1){
    printf("%d\n",i);
    if (i < 5){
      break;
    }
}
```

Output:

```
10
9
8
7
6
5
4
```

`continue` Keyword: myContinue.c

- `continue` is used to skip the rest of the commands in the loop and start from the top again
- The loop variable must still be incremented though

```
#include <stdio.h>
int main(){
   int i;
   i = 0;
   while ( i < 20 ){
      i++;
      continue;
      printf("Nothing to see\n");
}
   return 0;
}
```

The printf statement is skipped, therefore no output on screen.

References

- C Programming Language, Brian Kernighan and Dennis Ritchie
- Let Us C, Yashavant Kanetkar
- C for Dummies, Dan Gookin
- http://cplusplus.com
- http://www.cprogramming.com/tutorial/c/lesson11.html