Control Statements and Repetition

Jamal Hussein

jah1g12@ecs.soton.ac.uk

Electronics and Computer Science

November 5, 2015

UNIVERSITY OF Southampton

Outline

Constants

- There are two simple ways in C to define constants:
- Using `#define` preprocessor

```
#define identifier value
```

- Using `const` keyword

```
const type variable = value;
```

Constants

```c
#include <stdio.h>

#define PI 3.14159265359

int main()
{
        double area;
        double radius = 5;

        area = radius * radius * PI;
        printf("value of area : %f\n", area);

        return 0;
}
```

Constants

```c
#include <stdio.h>

int main()
{
        const double PI = 3.14159265359;
        double area;
        double radius = 5;

        area = radius * radius * PI;
        printf("value of area : %f\n", area);

        return 0;
}
```

Switch Statement

- A switch statement allows a variable to be tested for equality against a list of values

```
switch(expression){
        case constant-expression:
                statement(s);
                break; /* optional */
        case constant-expression:
                statement(s);
                break; /* optional */
        /* you can have any number of case statements */
        default : /* Optional */
                statement(s);
}
```

Switch Statement

- The expression used in a switch statement must have an integral or enumerated type
- The constant-expression for a case must be the same data type as the variable in the switch, and it must be a constant or a literal
- case will execute until a break statement is reached
- When a break statement is reached, the switch terminates
- If no break appears, the flow of control will fall through to subsequent cases
- No break is needed in the default case

Switch Statement

```
#include <stdio.h>
int main () {
        char grade = 'B';
        switch(grade) {
                case 'A' : printf("Excellent!\n" );
                           break;
                case 'B' :
                case 'C' : printf("Well done\n" );
                           break;
                case 'D' : printf("You passed\n" );
                           break;
                case 'F' : printf("Better try again\n" );
                           break;
                default : printf("Invalid grade\n" );
        }
        printf("Your grade is %c\n", grade );
        return 0;
}
```

For Statement

- A for loop is a repetition control structure that executes a specific number of times

```
for ( init; condition; increment)
{
        statement(s);
}
```

- The init step is executed first, and only once
- the condition is evaluated
  - If it is true (nonzero), the body of the loop is executed
  - if it is false (zero) the loop ended
- the flow of control jumps back up to the increment

For Statement

```
#include <stdio.h>
int main ()
{
        /* for loop execution */
        int a;
        for( a = 10; a < 20; a++ )
        {
                printf("value of a: %d\n", a);
        }
        return 0;
}
```

While Statement

- A while loop statement in C programming language repeatedly executes a target statement as long as a given condition is true

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

While Statement

```
#include <stdio.h>
int main ()
{
int a = 10;

/* while loop execution */
while( a < 20 )
{
printf("value of a: %d\n", a);
a++;
}

return 0;
}
```

Do-While Statement

- Unlike for and while loops, the do...while loop checks its condition at the bottom of the loop
- A do...while loop is guaranteed to execute at least one time

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

Do-While Statement

- Unlike for and while loops, the do...while loop checks its condition at the bottom of the loop
- A do...while loop is guaranteed to execute at least one time

```
#include <stdio.h>
int main ()
{
        int a = 10;
        /* do loop execution */
        do
        {
                printf("value of a: %d\n", a);
                a = a + 1;
        } while( a < 20 );
        return 0;
}
```

break statement

- When the break statement is encountered inside a loop, the loop is immediately terminated

```
#include <stdio.h>
int main ()
{
        int a = 10;
        /* while loop execution */
        while( a < 20 )
        {
                printf("value of a: %d\n", a);
                a++;
                if( a > 15)
                        break;
        }
        return 0;
}
```

continue statement

- continue statement forces the next iteration of the loop to take place, skipping any code in between

```
#include <stdio.h>
int main ()
{
        int a = 10;
        do
        {
                if( a == 15)
                {
                        a = a + 1;
                        continue;
                }
                printf("value of a: %d\n", a);
                a++;
        } while( a < 20 );
        return 0;
}
```

Nested Loops

The syntax of nested **for** loop

```
for ( init; condition; increment )
{
        for ( init; condition; increment )
        {
                statement(s);
        }
        statement(s);
}
```

Nested Loops

The syntax of nested **while** loop

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

Nested Loops

The syntax of nested **do-while** loop

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

Nested Loops

```
#include <stdio.h>
int main ()
{
        int i, j;
        for(i=2; i<100; i++) {
                for(j=2; j <= (i/j); j++)
                        if(!(i%j)) break;
                if(j > (i/j))
                printf("%d is prime\n", i);
        }
        return 0;
}
```