Introduction to Programming

C – Control Flow, Functions, Basic Input/output

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

December, 2020

Conditional – if-else

```
if(Condition){
    statement 1;
    statement 2;
}
else{
    statement 3; // Execute if Condition fails
}
```

Conditional – if-else

```
if(Condition){
    statement 1;
    statement 2;
}
else{
    statement 3; // Execute if Condition fails
}
```

Note: A single statement can be included (within if/else) by default without using any brackets.

Conditional – if-else

What will be the output of the following program?

```
if(!printf("Hi!"))
    printf("Hi!");
else
    printf("Bye!");
```

Conditional – if-else

What will be the output of the following program?

```
if(!printf("Hi!"))
    printf("Hi!");
else
    printf("Bye!");
```

Hi!Bye!

Conditional – switch-case

```
switch (E) {
    case value1 :
       statement;
       break;
    case value2 :
       statement;
       break;
    ...
    case valuen :
       statement;
       break;
    default:
       statement;
 }
```

Iterative – for loop

```
for(Initialization;Condition;Increment/Decrement){
    statement 1;
    statement 2;
}
```

Iterative – for loop

```
for(Initialization;Condition;Increment/Decrement){
    statement 1;
    statement 2;
}
```

Alternatively, we can write

```
for(Initialization;Condition;Increment/Decrement)
    statement 3;
```

Iterative – for loop

```
for(Initialization;Condition;Increment/Decrement){
    statement 1;
    statement 2;
}
```

Alternatively, we can write

```
for(Initialization;Condition;Increment/Decrement)
    statement 3;
```

Note: The 'Initialization', 'Condition', and 'Increment/Decrement' are all optional.

Iterative – for loop

What will be the output of the following program?

```
unsigned char i = 0;
for(;i>=0;i++);
    printf("%d\n",i);
```

Iterative – for loop

What will be the output of the following program?

```
unsigned char i = 0;
for(;i>=0;i++);
    printf("%d\n",i);
```

The program iterates infinitely!!!

Iterative – for loop

What will be the output of the following program?

```
unsigned char i = 0;
for(;i>=0;i++);
    printf("%d\n",i);
```

The program iterates infinitely!!!

Unsigned characters can never be negative and so $i >= 0$ never yields FALSE.

Iterative – while loop

```
while(Condition){
    statement 1;
    statement 2;
}
```

Iterative – while loop

```
while(Condition){
    statement 1;
    statement 2;
}
```

Alternatively, we can write

```
while(Condition)
    statement 3;
```

Iterative – while loop

```
while(Condition){
    statement 1;
    statement 2;
}
```

Alternatively, we can write

```
while(Condition)
    statement 3;
```

Note: A non-zero value is treated as a TRUE 'Condition', otherwise FALSE.

Iterative – do-while loop

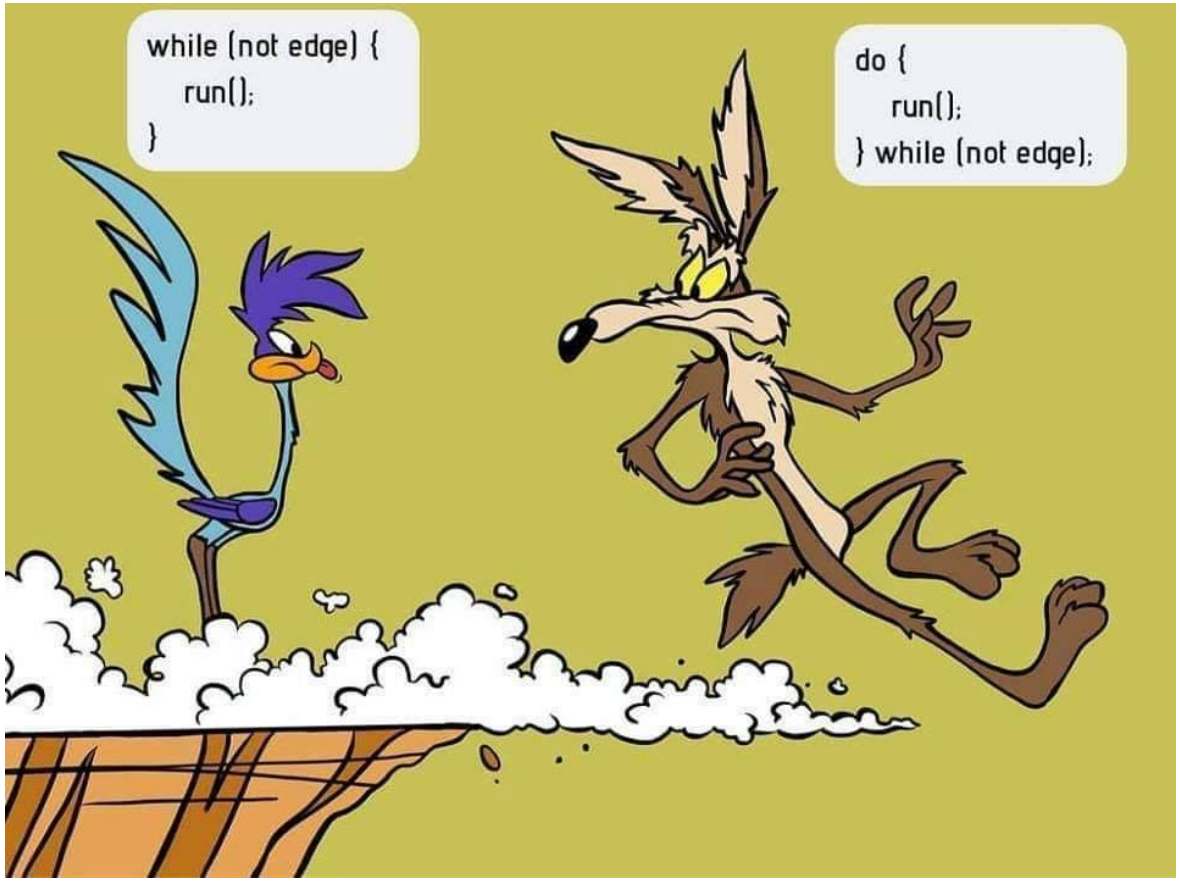
```
do{
    statement 1;
    statement 2;
}while(Condition);
```

Iterative – do-while loop

```
do{
    statement 1;
    statement 2;
}while(Condition);
```

Note: The statements inside the block are executed (by default) at least once irrespective of the 'Condition'.

Iterative – Comparing while and do-while



break and continue

- **break:** immediately jump to the next operation after the loop
- **continue:** do the update operation if applicable, and continue with the next iteration of the loop

break and continue

- **break:** immediately jump to the next operation after the loop
- **continue:** do the update operation if applicable, and continue with the next iteration of the loop

```
for(i=1; i<=100; ++i){
      printf("%4d",i);
      if (i%10 != 0)
         break;
      printf("\n");
}
```

break and continue

- **break:** immediately jump to the next operation after the loop
- **continue:** do the update operation if applicable, and continue with the next iteration of the loop

```
for(i=1; i<=100; ++i){
      printf("%4d",i);
      if (i%10 != 0)
         break;
      printf("\n");
}
```

```
i = 0;
while(i < 100){
      ++i;
      printf("%4d",i);
      if (i%10 != 0)
          continue;
      printf("\n");
}
```

Functions – Prototype declaration, definition, call

```
int max(int, int, int); // Prototype declaration

int max(int a, int b, int c){ // Definition starts
    return (a>b)?((a>c)?a:c):((b>c)?b:c);
} // Definition ends

int main(){
    int x = 7, y = 13, z = 11, maximum;
    maximum = max(x, y, z); // Call
    return 0;
}
```

Functions – Prototype declaration, definition, call

```
int max(int, int, int); // Prototype declaration

int max(int a, int b, int c){ // Definition starts
    return (a>b)?((a>c)?a:c):((b>c)?b:c);
} // Definition ends

int main(){
    int x = 7, y = 13, z = 11, maximum;
    maximum = max(x, y, z); // Call
    return 0;
}
```

- Calling function: `main()`
- Called function: `max()`
- Parameters: a, b, c (declared in `max()`)
- Arguments: x, y, z (passed to `max()`)

Functions – Efficient use

```
int max(int a, int b){
    if(a > b) {return a;}
    else {return b;}
}
```

Functions – Efficient use

```
int max(int a, int b){
    if(a > b) {return a;}
    else {return b;}
}

int max(int a, int b, int c){
    return max(max(a, b), c);
}
```

Inline functions

```
int main(){
    int x = 7, y = 13, maximum;
    inline int max(int x, int y) {return x > y ? x : y;}
    maximum = max(2, 3);
    return 0;
}
```

Inline functions

```
int main(){
    int x = 7, y = 13, maximum;
    inline int max(int x, int y) {return x > y ? x : y;}
    maximum = max(2, 3);
    return 0;
}
```

Note: Inline functions are compiled just as a regular code, but can be then inserted into compiled code and optimized as needed.

Input/Output

I/O from the terminal

- **printf, scanf**
- **getchar, putchar**

Input/Output

I/O from the terminal

- **printf**, **scanf**
- **getchar**, **putchar**

I/O from files

- File pointers: `FILE *`
- **fopen**, **fclose**
- **fprintf**, **fscanf**
- **fgetc**, **fputc**
- **fgets**, **fputs**

Input/Output

I/O from the terminal

- **printf**, **scanf**
- **getchar**, **putchar**

I/O from files

- File pointers: `FILE *`
- **fopen**, **fclose**
- **fprintf**, **fscanf**
- **fgetc**, **fputc**
- **fgets**, **fputs**

- Practice reading man pages.
  e.g. `$ man fopen`
- **Do not use** `gets()`**!**

Format specifiers

Specifying the format of a variable in the formatted Input/Output (printf()/scanf()) statement.

Format specifier	Purpose
%c	character I/O
%s	string/series of characters I/O
%d	signed integer I/O (assumes base 10)
%i	signed integer I/O (auto detects base)
%u	unsigned integer I/O
%o	unsigned octal integer I/O
%x/%X	unsigned hexadecimal integer I/O
%e/%E/%f/%g	double/floating point I/O
%lf	long double I/O

More on `scanf()`

We can take input strings with delimeters (e.g., spaces or commas) in between as follows.

```
char str[20];
scanf("%[^\t\n]s", str);
```

Command line arguments

The `main()` function can receive arguments directly from the command line as follows.

```
#include<stdio.h>
int main(int argument_count, char *argument_value[]){
    int i;
    for(i=0; i<argument_count; i++){
        printf("%s\n",argument_value[i]);
    }
    return 0;
}
```

Note: `argument_count` keeps the number of arguments passed from the command line and `argument_value[]` stores the arguments as array of strings. Both these variable names are user-defined.

Command line arguments

Execution:

```
./a.out DFS is fun
```

Output:

```
./a.out
DFS
is
fun
```