C: an introduction

Control flow: Loops

1

Program: building blocks

- Variables
  - Store data (input, intermediate values, results)
- Expressions
  - Manipulate variables
- Control structures
  - Make decisions (if) or repeat (for, while) statements
- Functions
  - Combine expressions and structures for parameterization and re-use

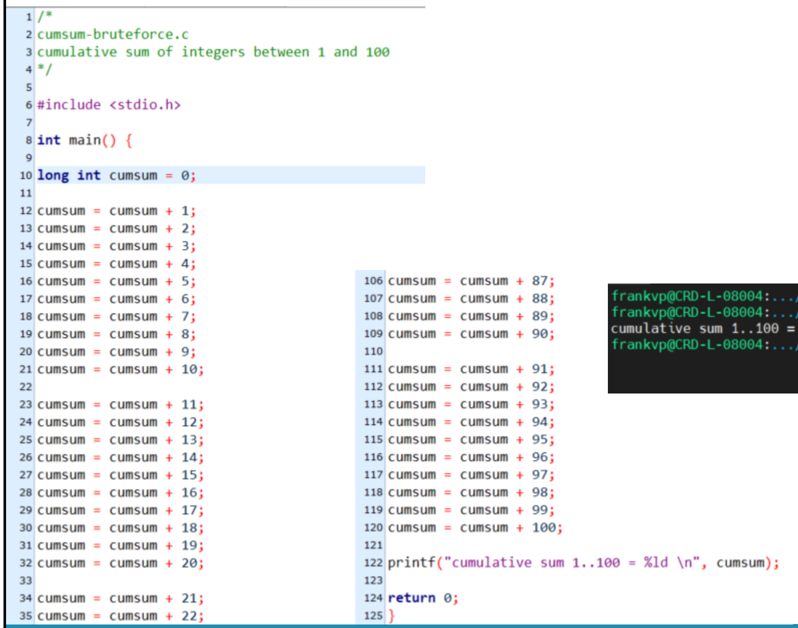
2

Why?

- Compute the cumulative sum of integers between 1 and 100
- Possible solution:

```
int cumsum = 0;
cumsum = cumsum + 1;
cumsum = cumsum + 2;
cumsum = cumsum + 3;
...
cumsum = cumsum + 100;
```

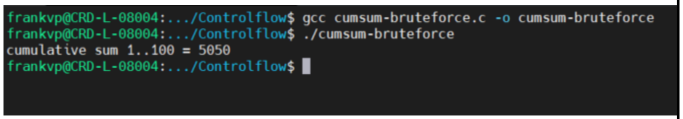
(http://gribblelab.org/cbootcamp/4_Control_Flow.html)



```c
/*
cumsum-bruteforce.c
cumulative sum of integers between 1 and 100
*/

#include <stdio.h>

int main() {

long int cumsum = 0;

cumsum = cumsum + 1;
cumsum = cumsum + 2;
cumsum = cumsum + 3;
cumsum = cumsum + 4;
cumsum = cumsum + 5;
cumsum = cumsum + 6;
cumsum = cumsum + 7;
cumsum = cumsum + 8;
cumsum = cumsum + 9;
cumsum = cumsum + 10;

cumsum = cumsum + 11;
cumsum = cumsum + 12;
cumsum = cumsum + 13;
cumsum = cumsum + 14;
cumsum = cumsum + 15;
cumsum = cumsum + 16;
cumsum = cumsum + 17;
cumsum = cumsum + 18;
cumsum = cumsum + 19;
cumsum = cumsum + 20;

cumsum = cumsum + 21;
cumsum = cumsum + 22;
```

```c
cumsum = cumsum + 87;
cumsum = cumsum + 88;
cumsum = cumsum + 89;
cumsum = cumsum + 90;

cumsum = cumsum + 91;
cumsum = cumsum + 92;
cumsum = cumsum + 93;
cumsum = cumsum + 94;
cumsum = cumsum + 95;
cumsum = cumsum + 96;
cumsum = cumsum + 97;
cumsum = cumsum + 98;
cumsum = cumsum + 99;
cumsum = cumsum + 100;

printf("cumulative sum 1..100 = %ld \n", cumsum);

return 0;
}
```

- *File: cumsum-bruteforce.c*



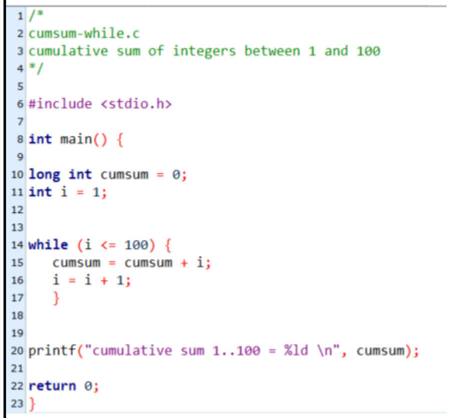
```
frankvp@CRD-L-08004:.../Controlflow$ gcc cumsum-bruteforce.c -o cumsum-bruteforce
frankvp@CRD-L-08004:.../Controlflow$ ./cumsum-bruteforce
cumulative sum 1..100 = 5050
frankvp@CRD-L-08004:.../Controlflow$
```

`while` statement

- the **`while`** loop has the form

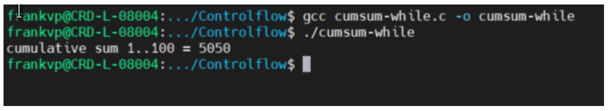
```
while (expression)
{
      statements;
}
```

- A **`while`** loop is executed *zero* or more times
- Should be used when the number of iterations is not known in advance



```c
/*
cumsum-while.c
cumulative sum of integers between 1 and 100
*/

#include <stdio.h>

int main() {

long int cumsum = 0;
int i = 1;


while (i <= 100) {
   cumsum = cumsum + i;
   i = i + 1;
   }


printf("cumulative sum 1..100 = %ld \n", cumsum);

return 0;
}
```

- *File: cumsum-while.c*



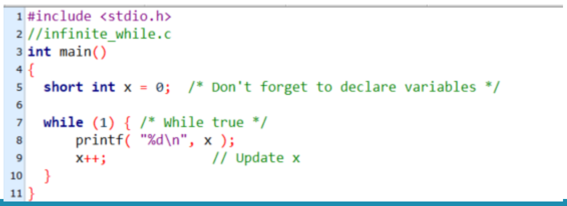
```
frankvp@CRD-L-08004:.../Controlflow$ gcc cumsum-while.c -o cumsum-while
frankvp@CRD-L-08004:.../Controlflow$ ./cumsum-while
cumulative sum 1..100 = 5050
frankvp@CRD-L-08004:.../Controlflow$
```

`while` statement

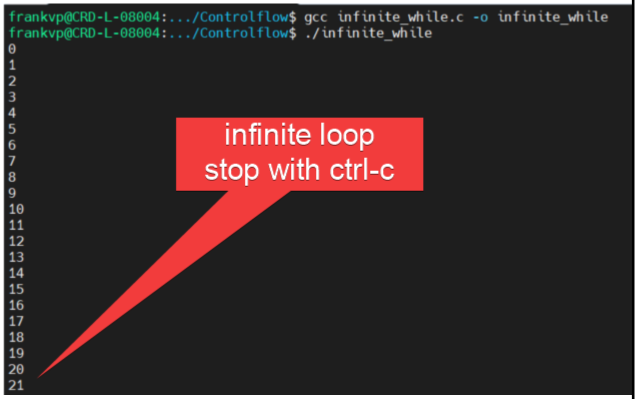
- infinite loop

```
while(1){
      do_something();
 }
```

- *File: infinite_while.c*



```c
#include <stdio.h>
//infinite_while.c
int main()
{
  short int x = 0;  /* Don't forget to declare variables */

  while (1) { /* While true */
      printf( "%d\n", x );
      x++;                 // Update x
  }
}
```



```
frankvp@CRD-L-08004:.../Controlflow$ gcc infinite_while.c -o infinite_while
frankvp@CRD-L-08004:.../Controlflow$ ./infinite_while
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

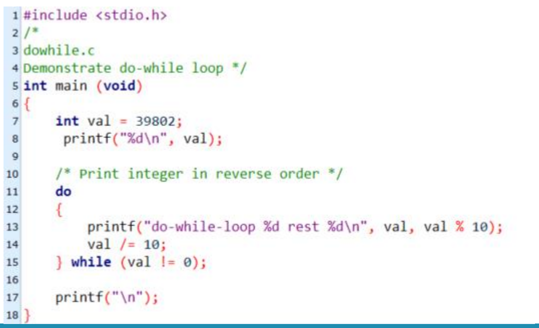
`do-while` statement

- the **`do-while`** loop has the form

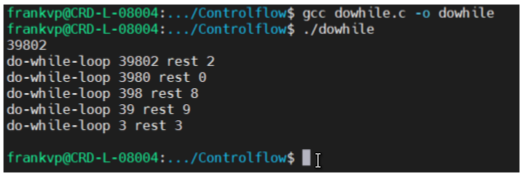
```
do
{
      statements
}
while (expression)
```

- A **`do-while`** loop is executed *one* or more times

- *File: dowhile.c*



```c
#include <stdio.h>
/*
dowhile.c
Demonstrate do-while loop */
int main (void)
{
    int val = 39802;
     printf("%d\n", val);

    /* Print integer in reverse order */
    do
    {
        printf("do-while-loop %d rest %d\n", val, val % 10);
        val /= 10;
    } while (val != 0);

    printf("\n");
}
```



```
frankvp@CRD-L-08004:.../Controlflow$ gcc dowhile.c -o dowhile
frankvp@CRD-L-08004:.../Controlflow$ ./dowhile
39802
do-while-loop 39802 rest 2
do-while-loop 3980 rest 0
do-while-loop 398 rest 8
do-while-loop 39 rest 9
do-while-loop 3 rest 3

frankvp@CRD-L-08004:.../Controlflow$
```

ICTS KU LEUVEN

`for` statement

- the `for` loop

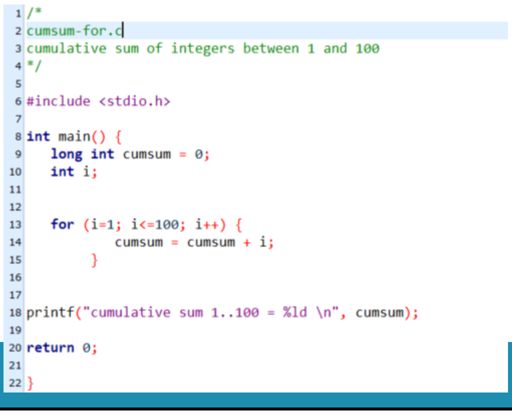
```
for (init; test; action)
    statement;
```

- Should be used when the number of iterations is known or computed
- Behaves exactly the same as

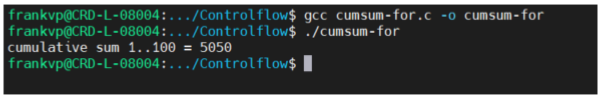
```
init;
while (test)
{
    statement;
    action;
}
```

KU LEUVEN

- *File: cumsum-for.c*



```
/*
cumsum-for.c
cumulative sum of integers between 1 and 100
*/

#include <stdio.h>

int main() {
   long int cumsum = 0;
   int i;


   for (i=1; i<=100; i++) {
         cumsum = cumsum + i;
       }


printf("cumulative sum 1..100 = %ld \n", cumsum);

return 0;

}
```



```
frankvp@CRD-L-08004:.../Controlflow$ gcc cumsum-for.c -o cumsum-for
frankvp@CRD-L-08004:.../Controlflow$ ./cumsum-for
cumulative sum 1..100 = 5050
frankvp@CRD-L-08004:.../Controlflow$ 
```

ICTS KU LEUVEN

`for` Loop Variants

- Multiple indices (uses the comma operator)

```
for(i=0, j=100; i <10; i++, j--){
     <statements>;
}
```

- Infinite loop

```
for( ; ; ){
     <statements>;
}
```

KU LEUVEN

Nested constructs

- As with **`if`**, the **`switch`**, **`while`**, **`do-while`**, and **`for`** constructs can all be nested.

```
for(expression){
  while(expression){
        if(expression){
                switch(int expression){
                case A: statements
                case B: statements
                }
…
```

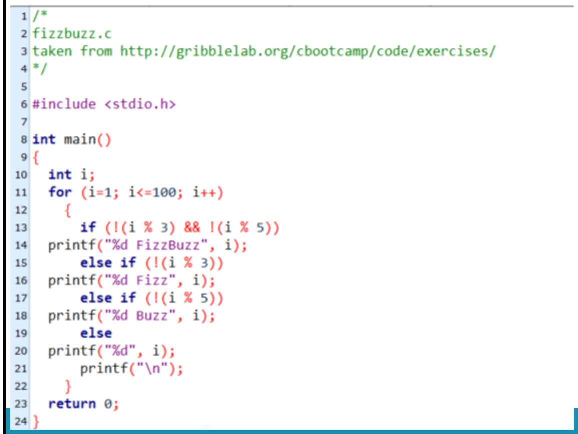
KU LEUVEN

Hands-on

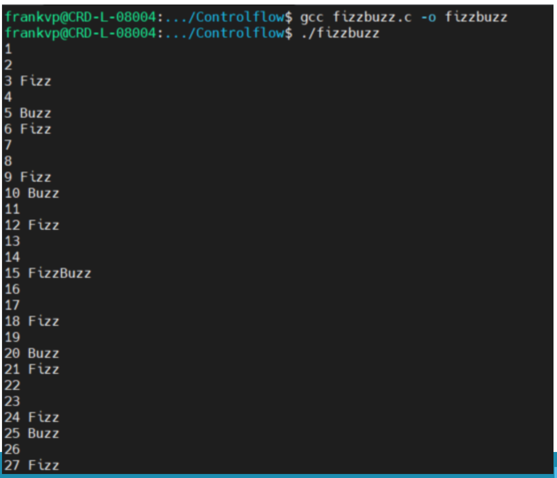
- Write a program that prints the numbers from 1 to 100.
  - For multiples of three print "Fizz" instead of the number
  - For the multiples of five print "Buzz".
  - For numbers which are multiples of both three and five print "FizzBuzz".

KU LEUVEN

File: fizzbuzz.c



```c
/*
fizzbuzz.c
taken from http://gribblelab.org/cbootcamp/code/exercises/
*/

#include <stdio.h>

int main()
{
  int i;
  for (i=1; i<=100; i++)
    {
      if (!(i % 3) && !(i % 5))
  printf("%d FizzBuzz", i);
      else if (!(i % 3))
  printf("%d Fizz", i);
      else if (!(i % 5))
  printf("%d Buzz", i);
      else
  printf("%d", i);
      printf("\n");
    }
  return 0;
}
```

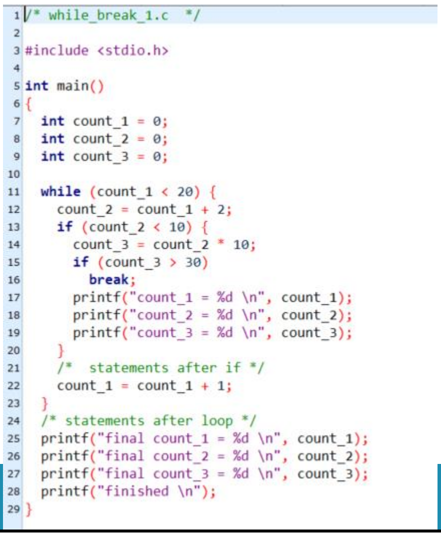


```
frankvp@CRD-L-08004:.../Controlflow$ gcc fizzbuzz.c -o fizzbuzz
frankvp@CRD-L-08004:.../Controlflow$ ./fizzbuzz
1
2
3 Fizz
4
5 Buzz
6 Fizz
7
8
9 Fizz
10 Buzz
11
12 Fizz
13
14
15 FizzBuzz
16
17
18 Fizz
19
20 Buzz
21 Fizz
22
23
24 Fizz
25 Buzz
26
27 Fizz
```

`break` statement

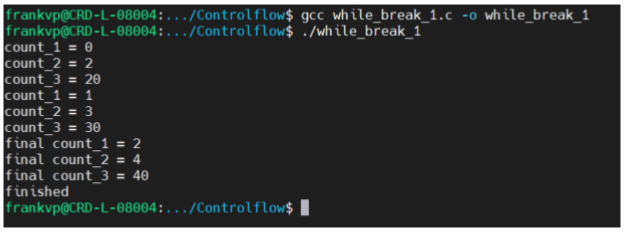
- the **`break`** statement causes transfer of control to the first statement *following* the innermost enclosing **`while`**, **`do`** or **`for`** loop, or **`switch`** statement.

```
while(expression){
   statement;
   statement;
   if(condition)
      break;
   statement;
   statement;
}
//control jumps here on break.
```



```c
/* while_break_1.c  */

#include <stdio.h>

int main()
{
  int count_1 = 0;
  int count_2 = 0;
  int count_3 = 0;

  while (count_1 < 20) {
    count_2 = count_1 + 2;
    if (count_2 < 10) {
      count_3 = count_2 * 10;
      if (count_3 > 30)
        break;
      printf("count_1 = %d \n", count_1);
      printf("count_2 = %d \n", count_2);
      printf("count_3 = %d \n", count_3);
    }
    /*  statements after if */
    count_1 = count_1 + 1;
  }
  /* statements after loop */
  printf("final count_1 = %d \n", count_1);
  printf("final count_2 = %d \n", count_2);
  printf("final count_3 = %d \n", count_3);
  printf("finished \n");
}
```

- *File: while_break_1.c*



```
frankvp@CRD-L-08004:.../Controlflow$ gcc while_break_1.c -o while_break_1
frankvp@CRD-L-08004:.../Controlflow$ ./while_break_1
count_1 = 0
count_2 = 2
count_3 = 20
count_1 = 1
count_2 = 3
count_3 = 30
final count_1 = 2
final count_2 = 4
final count_3 = 40
finished
frankvp@CRD-L-08004:.../Controlflow$ 
```

KU LEUVEN

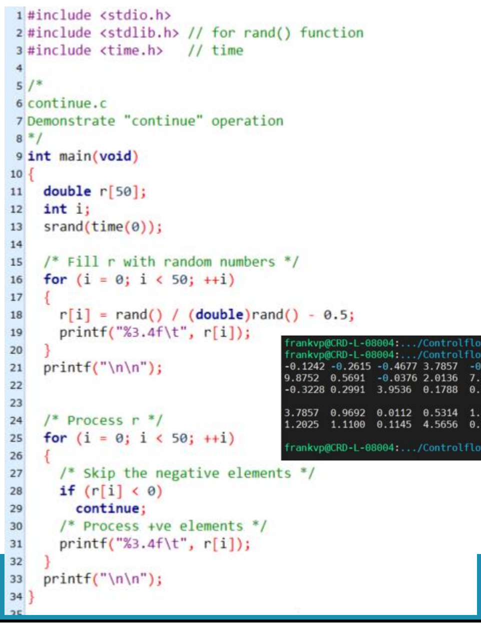
`continue` statement

- The **`continue`** statement causes transfer of control to the *beginning* of the innermost enclosing **`while`**, **`do`** or **`for`** loop. In the case of the **`for`** loop, the increment expression is executed first.

```
            while(expression){
               statement;
               if(condition)
                 continue;
               statement;
               statement;
               //control jumps here on continue
      }
```

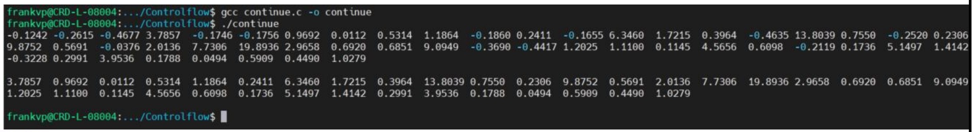
- Execution of the loop may continue following re-evaluation of the loop continuation condition test

KU LEUVEN



```c
#include <stdio.h>
#include <stdlib.h> // for rand() function
#include <time.h>   // time

/*
continue.c
Demonstrate "continue" operation
*/
int main(void)
{
  double r[50];
  int i;
  srand(time(0));

  /* Fill r with random numbers */
  for (i = 0; i < 50; ++i)
  {
    r[i] = rand() / (double)rand() - 0.5;
    printf("%3.4f\t", r[i]);
  }
  printf("\n\n");


  /* Process r */
  for (i = 0; i < 50; ++i)
  {
    /* Skip the negative elements */
    if (r[i] < 0)
      continue;
    /* Process +ve elements */
    printf("%3.4f\t", r[i]);
  }
  printf("\n\n");
}
```

- *File: continue.c*



```
frankvp@CRD-L-08004:.../Controlflow$ gcc continue.c -o continue
frankvp@CRD-L-08004:.../Controlflow$ ./continue
-0.1242 -0.2615 -0.4677 3.7857  -0.1746 -0.1756 0.9692  0.0112  0.5314  1.1864  -0.1860 0.2411  -0.1655 6.3460  1.7215  0.3964  -0.4635 13.8039 0.7550  -0.2520 0.2306
9.8752  0.5691  -0.0376 2.0136  7.7306  19.8936 2.9658  0.6920  0.6851  9.0949  -0.3690 -0.4417 1.2025  1.1100  0.1145  4.5656  0.6098  -0.2119 0.1736  5.1497  1.4142
-0.3228 0.2991  3.9536  0.1788  0.0494  0.5909  0.4490  1.0279

3.7857  0.9692  0.0112  0.5314  1.1864  0.2411  6.3460  1.7215  0.3964  13.8039 0.7550  0.2306  9.8752  0.5691  2.0136  7.7306  19.8936 2.9658  0.6920  0.6851  9.0949
1.2025  1.1100  0.1145  4.5656  0.6098  0.1736  5.1497  1.4142  0.2991  3.9536  0.1788  0.0494  0.5909  0.4490  1.0279

frankvp@CRD-L-08004:.../Controlflow$
```

loop guidelines

- a simple count-controlled loop: *for*-loop is the first choice
- an event-controlled loop, whose body has to be executed at least once, then a *do-while*-loop is appropriate
- an event-controlled loop, but nothing is known about the first execution, then a *while*-loop is appropriate