KU LEUVEN

C: an introduction

One More Thing

Union

- A union is just like a struct, except that instead of allocating space to store all the components, the compiler only allocates space to store the largest one, and makes all the components refer to the same address.
  - It is a collection of variables of different datatypes in the same memory location.
  - Define a union with many members, but at a given point of time only one member can contain a value.

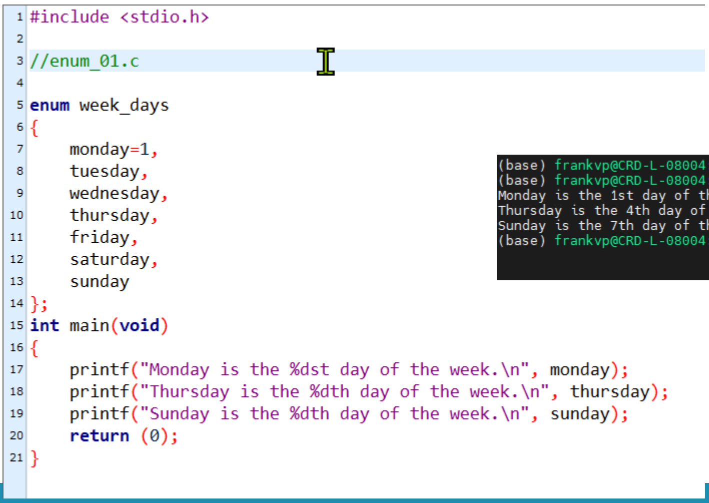
KU LEUVEN

Enumeration

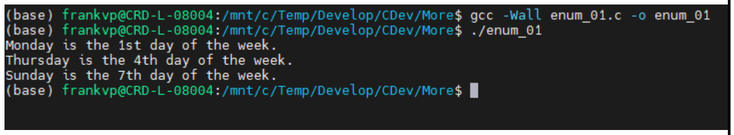
- Ordered list of all the items in a collection.
  - It particulary refers to a listing of all of the elements of a set. The enumerated types consists of integral constants and each integral constant is give a name.
  - Keyword `enum` is used to defined enumerated data type.
- Enums make your own type
  - Type is "list of key words"
  - Enums are useful for code clarity
  - Always possible to do the same thing with integers

KU LEUVEN

Enumeration



```c
#include <stdio.h>

//enum_01.c

enum week_days
{
    monday=1,
    tuesday,
    wednesday,
    thursday,
    friday,
    saturday,
    sunday
};
int main(void)
{
    printf("Monday is the %dst day of the week.\n", monday);
    printf("Thursday is the %dth day of the week.\n", thursday);
    printf("Sunday is the %dth day of the week.\n", sunday);
    return (0);
}
```

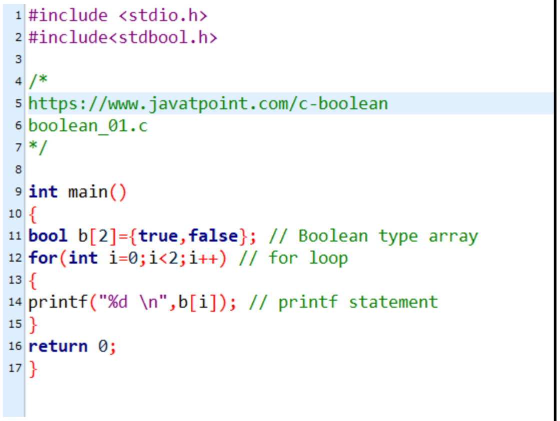


```
(base) frankvp@CRD-L-08004:/mnt/c/Temp/Develop/CDev/More$ gcc -Wall enum_01.c -o enum_01
(base) frankvp@CRD-L-08004:/mnt/c/Temp/Develop/CDev/More$ ./enum_01
Monday is the 1st day of the week.
Thursday is the 4th day of the week.
Sunday is the 7th day of the week.
(base) frankvp@CRD-L-08004:/mnt/c/Temp/Develop/CDev/More$
```

KU LEUVEN

boolean

- The C99 standard for C language supports bool variables
- Boolean is a data type that contains two types of values, i.e., 0 and 1. Basically, the bool type value represents two types of behavior, either true or false. Here, '0' represents false value, while '1' represents true value.
- Use the header file, `stdbool.h`.
- Syntax
  - `bool variable_name;`

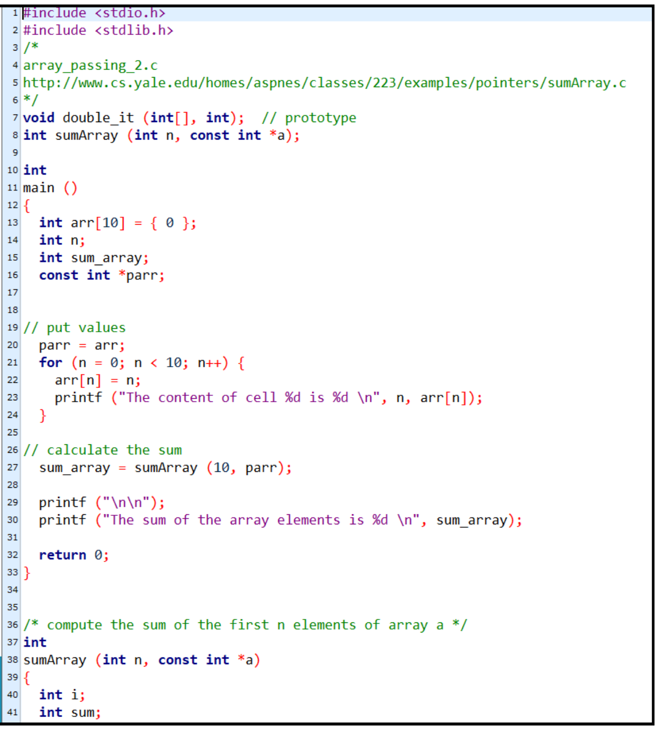


```c
#include <stdio.h>
#include<stdbool.h>

/*
https://www.javatpoint.com/c-boolean
boolean_01.c
*/

int main()
{
bool b[2]={true,false}; // Boolean type array
for(int i=0;i<2;i++) // for loop
{
printf("%d \n",b[i]); // printf statement
}
return 0;
}
```

KU LEUVEN

indent

- Command line tool changing the appearance of a C program by inserting or deleting whitespace.
- Can be used to make code easier to read. It can also convert from one style of writing C to another.
- `indent [options] [single-input-file] [-o output-file]`
- `indent -kr array_passing_2.c`



```c
#include <stdio.h>
#include <stdlib.h>
/*
array_passing_2.c
http://www.cs.yale.edu/homes/aspnes/classes/223/examples/pointers/sumArray.c
*/
void double_it (int[], int);  // prototype
int sumArray (int n, const int *a);

int
main ()
{
  int arr[10] = { 0 };
  int n;
  int sum_array;
  const int *parr;


// put values
  parr = arr;
  for (n = 0; n < 10; n++) {
    arr[n] = n;
    printf ("The content of cell %d is %d \n", n, arr[n]);
  }

// calculate the sum
  sum_array = sumArray (10, parr);

  printf ("\n\n");
  printf ("The sum of the array elements is %d \n", sum_array);

  return 0;
}


/* compute the sum of the first n elements of array a */
int
sumArray (int n, const int *a)
{
  int i;
  int sum;
```

splint

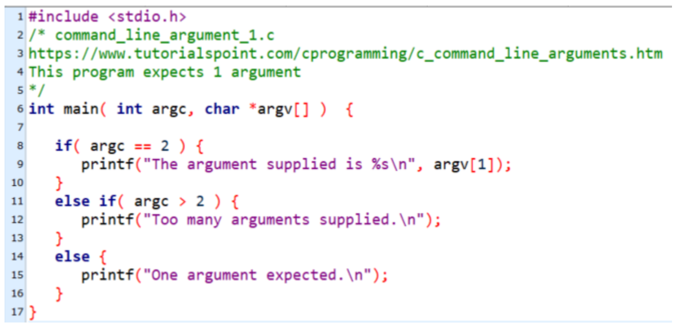
- A compiled C program is no guarantee that it will run correctly.
- The UNIX Lint tool Secure Programming Lint (SPLINT), can assist in checking for a multitude of programming errors.
- Check out man splint
- Run: `splint my_prog.c`
- Splint is particularly good at checking type checking of variable and function assignments, efficiency, unused variables and function identifiers, unreachable code and possible memory leaks.

KU LEUVEN

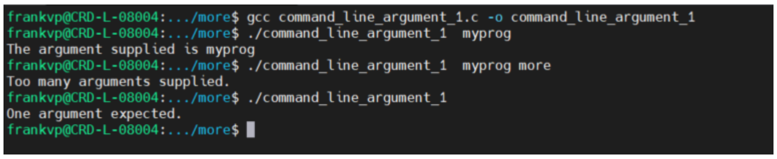
Command line arguments

- Pass some values from the command line to your C programs when they are executed: **command line arguments**
- The command line arguments are handled using `main()` function arguments
  - `argc` refers to the number of arguments passed,
  - `argv[]` is a pointer array which points to each argument passed to the program.
  - `argv[0]` holds the name of the program

ICTS KU LEUVEN



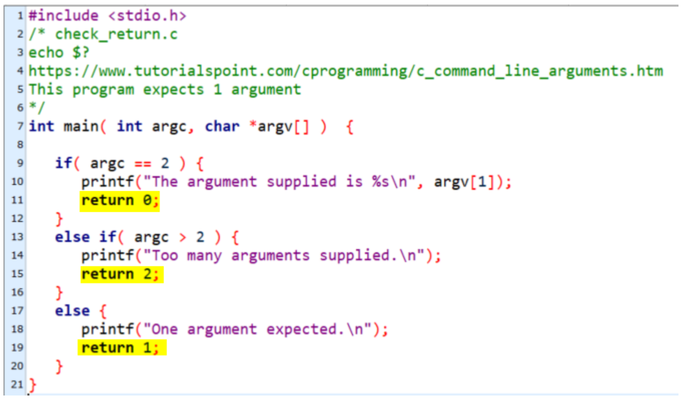
```c
#include <stdio.h>
/* command_line_argument_1.c
https://www.tutorialspoint.com/cprogramming/c_command_line_arguments.htm
This program expects 1 argument
*/
int main( int argc, char *argv[] )  {

   if( argc == 2 ) {
      printf("The argument supplied is %s\n", argv[1]);
   }
   else if( argc > 2 ) {
      printf("Too many arguments supplied.\n");
   }
   else {
      printf("One argument expected.\n");
   }
}
```



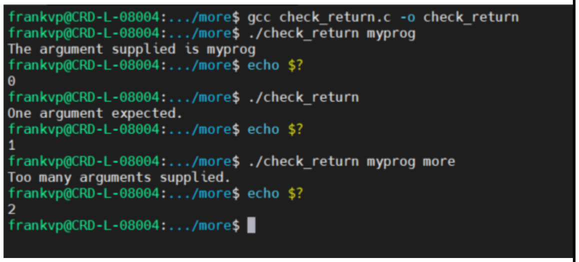
```
frankvp@CRD-L-08004:.../more$ gcc command_line_argument_1.c -o command_line_argument_1
frankvp@CRD-L-08004:.../more$ ./command_line_argument_1  myprog
The argument supplied is myprog
frankvp@CRD-L-08004:.../more$ ./command_line_argument_1  myprog more
Too many arguments supplied.
frankvp@CRD-L-08004:.../more$ ./command_line_argument_1
One argument expected.
frankvp@CRD-L-08004:.../more$
```

Return value

- Check the return value: `echo $?`



```c
#include <stdio.h>
/* check_return.c
echo $?
https://www.tutorialspoint.com/cprogramming/c_command_line_arguments.htm
This program expects 1 argument
*/
int main( int argc, char *argv[] )  {

   if( argc == 2 ) {
      printf("The argument supplied is %s\n", argv[1]);
      return 0;
   }
   else if( argc > 2 ) {
      printf("Too many arguments supplied.\n");
      return 2;
   }
   else {
      printf("One argument expected.\n");
      return 1;
   }
}
```



```
frankvp@CRD-L-08004:.../more$ gcc check_return.c -o check_return
frankvp@CRD-L-08004:.../more$ ./check_return myprog
The argument supplied is myprog
frankvp@CRD-L-08004:.../more$ echo $?
0
frankvp@CRD-L-08004:.../more$ ./check_return
One argument expected.
frankvp@CRD-L-08004:.../more$ echo $?
1
frankvp@CRD-L-08004:.../more$ ./check_return myprog more
Too many arguments supplied.
frankvp@CRD-L-08004:.../more$ echo $?
2
frankvp@CRD-L-08004:.../more$
```