KU LEUVEN

C: an introduction

One More Thing

Enumeration

- Ordered list of all the items in a collection.
  - Keyword `enum` is used to define a enumerated data type.
  - A list of named constant integer values (starting at 0 by default)
  - Each integral constant has a unique name.
  - Names in an enumeration must be distinct, but values need not.
- Enums make your own type
  - Consider type as “list of key words”
  - Enums can be useful for code clarity
  - Always possible to do the same thing with integers

KU LEUVEN

Enumeration

- Enumerations in C are numbers that have convenient names inside your code. They are not strings, and the names assigned to them in the source code are not compiled into your program, and so they are not accessible at runtime.
- The only way to get what you want is to write a function yourself that translates the enumeration value into a string.
- https://stackoverflow.com/questions/3168306/print-text-instead-of-value-from-c-enum

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$
```

union

- A special data type enabling you to store different data.
- 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

union

```
union_02.c
https://sceweb.sce.uncl.edu/helm/WEBPAGE-C/my_files/TableContents/Module-18/module18page.html


*/


int main( ) {

union Data {
   int i;
   float f;
   char str[20];
}; // a variable of Data type can store an integer, a floating-point number, OR a string

   union Data data;

   printf( "Memory size occupied by data : %ld\n", sizeof(data)); // memory sized for containing the largest member

   data.i = 10;
   printf( "data.i : %d\n", data.i);

   data.f = 220.5;
   printf( "data.f : %f\n", data.f);

   strcpy( data.str, "C Programming");
   printf( "data.str : %s\n", data.str);

   printf( "data.i : %d\n", data.i);
   printf( "data.f : %f\n", data.f);
   printf( "data.str : %s\n", data.str);
// values of i and f members of union got corrupted because the final value assigned to the variable has occupied the
 memory location

   return 0;
}
```

```
(base) frankvp@CRD-L-08004:.../CDev/More$ gcc union_02.c
(base) frankvp@CRD-L-08004:.../CDev/More$ ./a.out
Memory size occupied by data : 20
data.i : 10
data.f : 220.500000
data.str : C Programming
data.i : 1917853763
data.f : 41223605803277948604527599943 68.000000
data.str : C Programming
(base) frankvp@CRD-L-08004:.../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

Function pointer

- When calling a function with a function parameter, the value passed must be a pointer to a function.
  Use the function's name (without parentheses) for this

KU LEUVEN

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

/*
passing_function_01.c

basic example passing a function as a parameter to a function

based on https://stackoverflow.com/questions/9410/how-do-you-pass-a-function-as-a-parameter-in-c
*/

void print ( int x );
void func ( void (*f)(int) ); // prototype for a function which takes a function parameter

int main(void){

int a = 9;

print(a); // call the print function

func(print); // call func with the function print as a parameter

return EXIT_SUCCESS;

}

void print ( int x ) {
  printf("%d\n", x);
}

void func ( void (*f)(int) ) {
  for ( int ctr = 0 ; ctr < 5 ; ctr++ ) {
    (*f)(ctr);
  }
}
```

```
(base) frankvp@CRD-L-08004:/mnt/c/Temp/Develop/CDev/more$ gcc passing_function_01.c
(base) frankvp@CRD-L-08004:/mnt/c/Temp/Develop/CDev/more$ ./a.out
9
0
1
2
3
4
(base) frankvp@CRD-L-08004:/mnt/c/Temp/Develop/CDev/more$
```


assert

- Rather than using ‘if’ statements to check the return values,use the `assert()` function

```
#include <assert.h>
char *cp = malloc(22*sizeof(char));
assert(cp!=NULL);
```

- The parameter to assert is any Boolean expression- `assert(expression);`
  - If the Boolean expression is true, nothing happens and execution continues
  - If the Boolean expression is false, a message is output to stderr and the program terminates. The message includes the name of the .c file and the line number of the assert( ) that failed

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 (length of the array),
  - `argv[]` is a pointer array which points to each argument passed to the program.
  - `argv[0]` holds the name of the program


```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$
```

Command line arguments

- Common to use an array of pointers of type char*
- Often seen for parameters for `main` function

```
int main(int argc, char* argv[])
int main(int argc, char ** argv)
```


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 $?
2
frankvp@CRD-L-08004:.../more$
```

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.

lingo

- Definition >< declaration
  - A declaration provides basic attributes of a symbol: its type and its name.
  - A definition provides all of the details of that symbol—
    - if it's a function, what it does;
    - if it's a class, what fields and methods it has;
    - if it's a variable, where that variable is stored.
  - https://www.cprogramming.com/declare_vs_define.html#:~:text=Declaration%20vs%20Definition%3A%20In%20Summary,where%20that%20variable%20is%20stored.
  - https://stackoverflow.com/questions/1410563/what-is-the-difference-between-a-definition-and-a-declaration

lingo

- Expressions and statements
  - expressions are a "collection of symbols that make up a quantity" represents a single data item--usually a number.
    - a + b
    - t = u + v
    - x <= y
    - ++j
  - statements do something, cause the computer to carry out some definite action. There are three different classes of statements in C: expression statements, compound statements, and control statements.
    - An expression statement consists of an expression followed by a semicolon.
    - A compound statement consists of several individual statements enclosed within a pair of braces { }
  - https://farside.ph.utexas.edu/teaching/329/lectures/node11.html

Address space

- Taken from https://www.cl.cam.ac.uk/teaching/2223/ProgC/djg-primary-materials/c-notes-2223-djg-b.pdf
- A typical x86 32-bit address-space layout:

Description	Address
Top of address space	`0xffff ffff`
Stack (downwards-growing)	typical start `0x7fff ffff`
Heap (upwards-growing)	typical start `0x0020 0000`
Static variables	typical start `0x0010 0000`
C binary code	typical start `0x0000 8000`