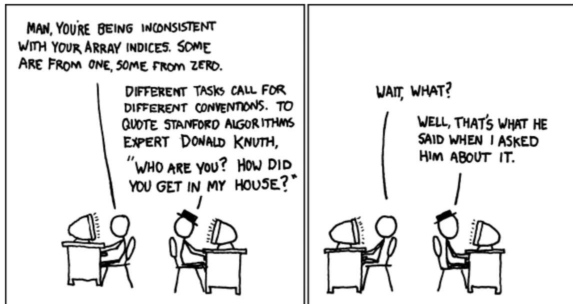
Week 4: Functions and Pointers and Characters

CS211: Programming and Operating Systems

Wednesday and Thursday, 03+04 March 2021



New class times

	Mon	Tue	Wed	Thu	Fri
9 – 10					
10 – 11					
11 – 12					
12 – 1					
1 – 2				***Zoom***	
2 – 3					
3 – 4	**LAB**		***Blackboard***		
4 – 5	**LAB**				

1. The recorded classes on Wednesdays and Thursdays are unchanged.
2. **New lab times: Monday 15:00-17:00**. You aim to attend for an hour. Drop in an out as needed.
3. Little, if any, of the “lab” times will be recorded.
4. Thursday, 4 March: **Will try Zoom:** `https://nuigalway-ie.zoom.us/j/92560272971?pwd=UFlGcHZwN2JkQXdocG1ZOG5HUFYrdz09` (Meeting ID: 925 6027 2971; Passcode: 465580)

This week, in CS211:

1. Part 1: Functions
   - Examples
   - `void`
2. Part 2: Call-by-value, and pointers
   - Pointers
3. Part 3: Characters
   - `03ASCII.c`
   - Important functions
4. Part 4: Strings in C
   - `string.h`
5. Part 5: Input and output of strings
   - Output
   - Input
6. Exercises

CS211
Week 4: Functions and Pointers and Characters

Start of ...

PART 1: Functions

Part 1: Functions

A good understanding of **functions**, and their uses, is of prime importance.

Some functions return/compute a single value.

However, many important functions return more than one value, or modify one of its own arguments. In these cases we need to know how to use pointers.

Part 1: Functions

Every C program has at least one function: `main()`

Example

```
#include <stdio.h>

int main(void )
{
   /* Stuff goes here */
   return(0);
}
```

Part 1: Functions

Each function consists of two parts:

- Function "header" or ***prototype*** which gives the function's
  - return value data type, or `void` if there is none, and
  - parameter list data types; or `void` if there are none.
  - The parameter list can, optionally, include variable names, but these are treated like comments, and ignored.

  The prototype is often given near the start of the file, before the ***main()*** section.

- ***Function definition***:
  - Begins with the function name, parameter list and return type,
  - followed by the body of the function contained within curly brackets.
  - Unless the return type is `void`, it ends with a `return`.

We will now look at three examples:

- computing the mean of two floats,
- compute the factorial of an int.
- compute the greatest common divisor of two (positive) integers.

00mean.c

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

   float mean(float, float);    // Prototype

   int main(void)
14 {
     float a, b;
16   printf("Enter (floating-point) numbers a and b: ");
     scanf("%f", &a);
18   scanf("%f", &b);
     printf("mean(a,b)=%f\n", mean(a,b));
20   return(0);
   }

   float mean(float a, float b)
24 {
     return( (a+b)/2.0);
26 }
```

01factorial.c

```
   int factorial(int n);   // Prototype

   int main(void)
14 {
     int x;
16   printf("Enter a positive integer:  ");
     scanf("%d", &x);   // Warning: should do input check
18   printf("factorial(%d)=%d\n",
            x, factorial(x));
20   return(0);
   }

   int factorial(int n)   /* Defination */
24 {
     int i, fac=1;
26   for (i=1; i<=n; i++)
       fac = fac*i;
28   return(fac);
   }
```

02gcd.c

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

int gcd(int a, int b);

int main(void)
{
  int a, b;
  printf("Enter a and b: ");
  scanf("%d", &a);
  scanf("%d", &b);
  printf("gcd(a,b)=%d\n", gcd(a,b));
  return(EXIT_SUCCESS);
}
```

02gcd.c

```
22 int gcd(int a, int b)
   {
24   int x=a, y=b, r;

26   while(y != 0)
     {
28     r = x%y;
       x=y;
30     y=r;
     }
32   return(x);
   }
```

Part 1: Functions

In three previous examples, the functions all took one or more arguments, and returned some value.

- Some functions return no values, so the return type is `void`;
- Some functions take no inputs, so the parameter list is `void`;

Example:

```
#include <stdio.h>
void Banner(void);

int main(void )
{
  /* ...  */
  Banner();
  /* ... */
}
```

```
void Banner(void )
{
  printf("\nThis is intro.c\n’’);
  printf("%s%s\n");
    "It prints this message",
    "when the program starts");
}
```

CS211
Week 4: Functions and Pointers and Characters

END OF PART 1

CS211
Week 4: Functions and Pointers and Characters

Start of ...

PART 2: Call-by-value and Pointers

Part 2: Call-by-value, and pointers

In C, it is ***very*** important to distinguish between

- a variable
- the value stored in it.

A good example is as follows: write a C function as follows:

- the function is called `Swap()`
- takes two integer inputs `a` and `b`
- after calling the function, the values of `a` and `b` are swapped.

Part 2: Call-by-value, and pointers

Call-By-Value.c

```
void Swap(int i, int j);

int main(void )
{
   int i, j;

   printf("Enter an integer: "); scanf("%d", &i);
   printf("Enter an integer: "); scanf("%d", &j);

   printf("i=%2d and j=%2d\n",i,j);
   printf("Swapping...\n");
   Swap(i,j);
   printf("i=%2d and j=%2d\n",i,j);
```

Part 2: Call-by-value, and pointers

```
void Swap(int a, int b)
{
  int tmp;

  tmp=a;
  a=b;
  b=tmp;
}
```

This won't work! We will only have passed the *values stored in the variables i and j*. even if these are swapped in the function, they remained unchanged in the calling function.

What we really wanted to do here was to use ***Call-By-Reference*** where we modify the contents of the memory space referred to by `i` and `j`.

A variable has a location in memory. The value of the variable is stored at that location. Example:

```
int i=10;
```

tells the system to allocate a location in memory for storing integers can be referred to as `i`. Furthermore, the value `10` should be stored there.

One of the distinguishing features of C is that we can manipulate the address of the variable almost as easily as changing its value.

The important concepts are

- if `i` is a variable, then `&i` is its location in memory.
- The declaration `int *p` creates a variable called `p` that can store the memory address of an integer.
- If a memory address is stored in the variable `p`, then `*p` is the value at that address.

The correct version of the `Swap` function and program is now:

Swap_by_Reference

```
void Swap_by_Reference(int *p, int *q)
{
  int tmp;
  tmp=*p;    *p=*q;    *q=tmp;
}
```

This is called as follows

From main

```
   printf("i=%2d and j=%2d\n",i,j);
   printf("Swapping...\n");
   Swap_by_Reference(&i,&j);
   printf("i=%2d and j=%2d\n",i,j);
```

CS211
Week 4: Functions and Pointers and Characters

END OF PART 2

CS211
Week 4: Functions and Pointers and Characters

Start of ...

PART 3: Characters

Part 3: Characters

In C, a `char`atacer is just an unsigned integer; it is how you use it that matters. Each character corresponds to an integer between 0 and 127.

What's so special about 127?

For example, the line

```
printf("%c == %c \n", 'a', 97);
```

will yield the output: *a == a*

Some ASCII codes are given below

32	48	57	65	90	97	122
`space`	`0`	`9`	`A`	`Z`	`a`	`z`

For more codes: see `03ASCII.c`

03ASCII.c

```
#include <stdio.h>

int main(void ) {
  int i, start, step=16;

  for (start=32; start < 127; start+=step)
    {
      printf("\n%12s", "Code:");
      for (i=start; i < start+step; i++)
        printf("%4i", i);

      printf("\n%12s", "Character:");
      for (i=start; i < start+step; i++)
        printf("%4c", i);
      printf("\n");
    }
  printf("\n");
  return(0);
}
```

- `printf("%c", c);` will send the character stored in `c` to the screen.
- `putchar(c);` same as above.
- `scanf("%c", &c);` will take a character form the keyboard input and stored it in `c`.
- `c = getchar();` ditto.

Example: Write a function that takes an character as input and, if that character is lower case, return the corresponding upper case character.

03uppitty.c

```
10 #include <stdio.h>

12 char upify(char c);

14 int main(void ) {
     char c;
16   while( (c=getchar()) != '\n')
         printf("upify( %c ) = %c \n", c, upify(c));
18   return(0);
   }

   char upify(char a)
22 {
     if ((a >= 'a') && (a <= 'z'))
24     return(a - 'a' + 'A');
     else
26     return(a);
   }
```

CS211
Week 4: Functions and Pointers and Characters

END OF PART 3

CS211
Week 4: Functions and Pointers and Characters

Start of ...

PART 4: Strings in C

Part 4: Strings in C

Now we will look at **strings**. Usually, these are thought of a collection of letters/characters that make up a word or a line of text.

The C language **does not actually have a `string` data type**. Instead, it uses arrays of type `char`.

If you make a declaration like:

```
char greeting[20]="Hello.  How are you?";
```

the system stores each character as an element of the array `greeting[]`.

Part 4: Strings in C

Some Important Points:

1. In the above example we declared the array to be of length 20. Even though the string contains 19 characters, an extra ***string termination character*** `\0` (backslash zero) is added to show where the end of the string is.
2. Spaces or even new-line characters do not terminate a string. They are treated just like other characters.
3. Declarations are the only time we can use an "equals" to assign a value to a string. At all other times, we can modify the string one character at a time:
   `greeting[0]='H'; greeting[1]='e';` ...
4. Better still use `strcpy()` – the "string copy" function:
   `strcpy(greeting, "Not too bad");`

Part 4: Strings in C

The `strcpy()` is one of a collection of functions for dealing with strings. Its definition is to be found in the `string.h` header file. More of this later...

Example: *Write a function that determines the length of a string.*

Part 4: Strings in C

05StringLength.c

```
#include <string.h> // Needed for strcpy

int length(char *);

int main(void )
12 {
   char greeting[20];
14   strcpy(greeting, "Hello. How are you?");
   printf("%s\n", greeting);
16   printf("That message was %d chars long.\n", length(greeting) );
   return(0);
18 }

20 int length(char *str)
{
22   int i, len=0;

24   for (i=0; str[i] != '\0'; i++)
     len++;

   return(len);
28 }
```

Useful functions defined in `string.h` include:

strncpy

```
char *strncpy(char *dest, const char *source, int n);
```

Copies at most `n` character from the string in `source` to `dest`. The advantage is that we won't copy more characters to `dest` than is allowed

Example

```
char Code[6], Name[20]="Operating Systems";
strcpy(Code, Name); // Bad!  Unexpected Results
strncpy(Code, Name, 6); // OK.
```

strcat

`strcat()`: Con**cat**enate two strings, i.e., append one string onto the end of another. E.g,

```
char message1[30]="Hello.";
char message2[30]=" How are you?";
strcat(message1, message2);
```

Now `message1` contains "Hello. How are you?";

strcmp

`strcmp(char *s1, char *s2)`: **Comp**ares two stings. It returns an integer:

- 0 if they are the same,
- negative if s_1 is the first alphabetically
- positive if s_2 comes first

Example

```
char Name0[20], Name1[20], First[20];
strncpy(Name0, "Richie", 20);\\
strncpy(Name1, "Dennis", 20);\\

if ( strcmp(Name0, Name1) > 0)
    strncpy(First, Name1, 20);
```

strlen

`strlen` Takes a single (pointer to) a string as its argument and returns an integer equal to its **len**gth minus 1. (**Why -1?**).

strstr

```
char *strstr( char *haystack, char *needle);
```

Searches for the first occurrence of the string `needle` in `haystack`. It returns a pointer to the start of the matching substring.
Moreover, if `needle` is ***not*** found in `haystack` it returns NULL.

Example:

```
if (strstr(Code, "CS") != NULL)
   printf("%s is a CS course\n", Code);
```

CS211
Week 4: Functions and Pointers and Characters

END OF PART 4

Start of ...

PART 5: Input and output of strings

You all know how to use `printf()` with strings:

```
printf("%s%s\n", "Good morning ", name);
```

or

```
printf("%s%8s\n", "Good morning ", name);
```

In the second example the *field width* specifier is given. This causes the second string to be "padded" so that it takes up a total of 8 spaces. This is useful for tabulated output.

One could also use `puts()`: this prints the contents of a string followed by a new-line character.

Input is a more complicated issue, but there are three basic methods:

- `scanf("%s", name);` reads a the next “word” from the input buffer (usually the key board) and stores it in the array `name[]`. A word is a sequence of characters that does not include a space, tab or newline character.
- to get more control of the input, you could use `getchar()` within a loop:

```
printf("What is your name? ");
for (i=0;
    (myname[i] = getchar()) != '\n';
    i++);
myname[i]='\0';
```

- `gets(string)`: this reads a line a input and stores it all (except the `'\n'`) in the array pointed to by `string`. This would be very useful, except that `gets()` is known to be buggy and is best avoided.

From the Linux manual page from `gets()`:

```
BUGS
Never use gets().  Because it is impossible to tell
without knowing the data in advance how many chars gets()
will read, and because gets() will continue to store
characters past the end of the buffer, it is extremely
dangerous to use.  It has been used to break computer
security.  Use fgets() instead.
```

- `fgets(string, n, stdin)`: reads in a line of text from the keyboard (standard input) and stores at most `n` characters in array `sting`. The new line charater is stored.

Which ever you use is a matter of choise. My preference is always to write functions that use `getchar()` and related functions, particularly if reading from a file.

Exercises

Exercise (Exer 4.1)

Write a short C programme that prompts the user to input an integer, and then uses `scanf` to read that integer.
The program should output the value that the user entered and that `scanf` returns.
Run the program to check what `scanf` will return when

(i) *the user enters an integer;*

(ii) *the user enters a float (with decimal part);*

(iii) *the user enters non-digit character.*

Exercises

Exercise (4.2)

The `uppitty` function in `02uppitty.c` is a bit trivial, not least because there is a C function, `toupper`, that already does this.
Write a variant as follows:

- *Its argument is a pointer to type character.*
- *the function **changes** the character to lower case.*
- *Write a similar function called `downify()` that converts an upper-case character to lower case, but leave all other characters unchanged.*

Exercises

Exercise (4.3)

On Twitter, the satirist John Bull (`@garius`) represents the words of "Gove" using a (seemingly random) mixture of upper- and lower-case text, as in this example from
https://twitter.com/garius/status/1090260422836477952

> *ThE sNOw gLOwS WhITe oVeR WhitEHall toNIGht*
> *nOT a sTAteSMan tO Be seEn UnITEd KinGDom ISolatED*
> *a PM wHo THInks sHe's a QUEen*

Write a function that takes a `char` array as an argument and "GoVEifies" it by changing letters to upper or lower case at random. You may use the built-in `tolower` and `toupper` functions.