15 Pointers, arrays, and strings

15.1 Pointers and arrays

- A *pointer variable* is one which contains the address of some data. An array is a restricted kind of pointer variable, because its value is an address.
- Asterisk denotes a 'pointer type'

```
int * x; // x is pointer to int
         // one expects that x contains the address
         // of some integer data.
```

- A pointer variable can be treated as an array.

```
int a[3] = {1,2,3};
      // a[0], a[1], a[2] are defined and initialised
int * x;
      // x[0], x[1], ... are defined all right, but
      // since x is not initialised, reference to
      // these items will probably cause the program
      // to crash.
char *y; // y is, or can be treated as, a character string.
char * argv[];
      // an array of character strings!
char * day[] = {"Sun", "Mon", ... };
      // an array of character strings.
```

- ```
  int a[3]={1,2,3};
  int * x = a;
        // This is correct. The value of x is the
        // address of a[0]; not only are x[0] and a[0]
        // equal (to 1), they occupy exactly the same
        // place in memory.
  ```

15.2 Allocation

It is possible to get blocks of memory in which to store data.

```
#include <stdlib.h>
char * x;
x = (char*) calloc ( 100, 1 );
      // allocates 100 bytes of memory and stores
      // its address in x.  The (char*) is called a
      // CAST. Without the cast, the
      // type of 'calloc' is 'pointer with unspecified type.'
      // The cast changes its type to
```
```

```
// 'pointer to char.'  I'm not sure that
// the cast is necessary.  With or without the cast,
// the value (an address) is the same.
// More about casts later.
```

15.3 Inputting text

There is an important constant in C, the 'null pointer.'

`NULL`
is a constant in C.

It is a 64-bit (or 32-bit) integer whose value is ***NOT*** the address of any data. (Actually, it is zero. It is defined in `stdio.h`.)

The function, and it is a function,

```
char* fgets ( char buffer[], int max_count, FILE * input_file )
```

does the following

- We only use 'standard input,' so for this module the input file is always `stdin`.
- If end-of-data has not been reached, it reads characters into the buffer (i.e., an array of characters) until either
  - a newline character has been read, or end-of-data is reached, or
  - `max_count - 1` characters have been read. This is to protect against overflowing the array `buffer`.
- and a null character `'\0'` is appended.
- It returns the *value* of `buffer`!
- At end-of-data, it returns `NULL`.

Application

```
while ( fgets ( buffer, 200,stdin ) != NULL )
{
  // etcetera
}
```

15.4 nostrings.c

Here is a program intended to read in several lines of text

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

void decr ( char * x )  // x is a char string.
                        // Usually fgets includes the
                        // newline '\n'.  This overwrites
                        // it with '\0', thus removing it.
{
  int i = 0;
  while ( x[i] != '\0' )
    if ( x[i] == '\n' )
      x[i] = '\0';
    else
      ++i;
}

int main ( )
{
  char buffer[200];
  char * line[1000];
  int count = 0;
  while ( fgets ( buffer, 200, stdin ) != NULL )
  {
    decr ( buffer ); // I prefer to delete the newlines.
    line[count] = buffer;
    ++ count;
  }

  int i;
  for (i=0; i<count; ++i)
  {
    printf("%s\n", line[i]);
  }
}
```

input:
We know that you highly esteem the kind of Learning taught in those Colleges, and that the Maintenance of our young Men, while with you, would be very expensive to you. We are convinc'd, therefore, that you mean to do us Good by your Proposal; and we thank you heartily. But you, who are wise, must know that different Nations have different Conceptions of things; and you will therefore not take it amiss, if our Ideas of this kind of Education happen not to be the same with yours. We have had some experience of it. Several of our young People were formerly brought up in the Colleges of the Northern

```
Provinces; they were instructed in all your Sciences; but, when

output:
Provinces; they were instructed in all your Sciences; but, when
Provinces; they were instructed in all your Sciences; but, when
Provinces; they were instructed in all your Sciences; but, when
Provinces; they were instructed in all your Sciences; but, when
Provinces; they were instructed in all your Sciences; but, when
Provinces; they were instructed in all your Sciences; but, when
Provinces; they were instructed in all your Sciences; but, when
Provinces; they were instructed in all your Sciences; but, when
Provinces; they were instructed in all your Sciences; but, when
Provinces; they were instructed in all your Sciences; but, when
```

15.5 strings.c

We need to copy the contents of the buffer to `line[]`, not its address. To do this we use the following functions:

```
  strlen ( char * x )  length of a character string
                (string.h)
  calloc ( int count, int size ) (stdlib.h)
                finds an unused block of count*size bytes
                of memory, cleared to zero, and returns
                its address.
  snprintf(char buffer[], int max_size, char* format,
                item_1, item_2, ...) (stdio.h)

                This is an 'internal' form of printf which
                stores in the buffer the same characters
                as printf would print to output.  BUT
                the size is a precaution against overflowing the
                buffer.

        We use this only to copy one string to another.

        char target_string[100];
        snprintf(target_string, 100, "%s", source_string);
```

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

void decr ( char * x )
{
```

```
  int i = 0;
  while ( x[i] != '\0' )
    if ( x[i] == '\n' )
      x[i] = '\0';
    else
      ++i;
}

int main ( )
{
  char buffer[200];
  char * line[1000];
  int count = 0;
  while ( fgets ( buffer, 200, stdin ) != NULL )
  {
    decr ( buffer );
                // delete newline, a matter of taste

    int size = 1 + strlen ( buffer );
                // note the extra 1 for the null character

    line[count] = calloc(1,size);
                // an unused block of memory

    snprintf(line[count],size,"%s",buffer);
                // this copies the buffer to line[count]
    ++ count;
  }

  int i;
  for (i=0; i<count; ++i)
  {
    printf("%s\n", line[i]);
  }
}

input: same as above

output:
We know that you highly esteem the kind of Learning taught in those
Colleges, and that the Maintenance of our young Men, while with
you, would be very expensive to you.  We are convinc'd, therefore,
that you mean to do us Good by your Proposal; and we thank you
heartily.  But you, who are wise, must know that different Nations
have different Conceptions of things; and you will therefore not take
```

it amiss, if our Ideas of this kind of Education happen not to be the same with yours. We have had some experience of it. Several of our young People were formerly brought up in the Colleges of the Northern Provinces; they were instructed in all your Sciences; but, when