Using Arrays in C

Video 19

Daniel J. Bodony
Department of Aerospace Engineering
University of Illinois at Urbana-Champaign

In this video you will learn...

1. About arrays in C
2. How to allocate memory

About Arrays in C

For engineers, arrays are a necessary item. We used them extensively in Matlab and could create and destroy them without thinking much. For example, this is simple code in Matlab to create two vectors, `x` and `f`:

```
>> x = linspace(0,1,100);
>> f = sin(2*pi*x);
```

How do we do this in C? There are two main ways: (1) static arrays and (2) dynamic arrays.

Static Arrays in C

The term static refers to how the memory is allocated; statically. In Matlab we never worried about memory; it was always "just there." In C (and it any compiled programming language) you must worry about memory.

In the example from the previous slide:

```
>> x = linspace(0,1,100);
>> f = sin(2*pi*x);
```

two arrays `x` and `f` are created to have 100 elements and to be of type `double`.

Static Arrays in C

The term static refers to how the memory is allocated; statically. In Matlab we never worried about memory; it was always "just there." In C (and it any compiled programming language) you must worry about memory.

In the example from the previous slide:

```
>> x = linspace(0,1,100);
>> f = sin(2*pi*x);
```

two arrays `x` and `f` are created to have 100 elements and to be of type `double`. Here's the C-version of the same code:

```
double x[100], f[100]; /* this is static allocation */
int i;
for (i = 0; i < 100; i++) {
  x[i] = double(i) / (100.0 - 1.0);
  f[i] = sin(2.0 * pi * x[i]);
}
```

Static Arrays in C

Let's break down this code bit-by-bit. The first line

```
double x[100], f[100]; /* this is static allocation */
```

declares two new arrays of type double be created with 100 elements. In C the counting starts from 0, so you'd access the array elements by `x[0]`, `x[1]`, and so on until `x[99]`. The element `x[100]` does not exist!

It is good programming practice to define the array size as a constant, like this

```
#define N 100
...
double x[N], f[N]; /* this is static allocation */
```

Static Arrays in C

Once we have the arrays declared, we can fill them up with this code:

```
int i;
for (i = 0; i < 100; i++) {
  x[i] = (double)(i) / (100.0 - 1.0);
  f[i] = sin(2.0 * pi * x[i]);
}
```

Notice that we must declare the counter, `i`, just like the arrays `x` and `f`. In C you must declare every variable!

Notice in the `for` loop the expression `i++`. This is a special C short-cut for `i = i+1`.

Static Arrays in C

The full program, say `static.c`, would look like this:

```
#include <stdio.h>
#include <stdlib.h>
#define N 100
int main (void) {
  double x[N], f[N];
  int i;
  for (i = 0; i < N; i++) {
    x[i] = (double)(i) / ((double)(N) - 1.0);
    f[i] = sin(2.0 * pi * x[i]);
  }
  return EXIT_SUCCESS;
}
```

Notice the use of the variable `N` instead of the number 100. It is always a good idea to avoid hardcoding the size of arrays and loop bounds.

Dynamic Arrays in C

If we compiled `static.c` and ran the code using the commands

```
$ gcc static.c -o static
$ ./static
```

we would always get the same output, with 100 points used for `x` and `f`. But what if we wanted to change to 200, or 2000, or some other number of points?

Dynamic Arrays in C

If we compiled `static.c` and ran the code using the commands

```
$ gcc static.c -o static
$ ./static
```

we would always get the same output, with 100 points used for `x` and `f`. But what if we wanted to change to 200, or 2000, or some other number of points?

Answer: you would need to change the value of `N` and then recompile. You could save different executables. But this isn't very efficient.

Dynamic Arrays in C

If we compiled `static.c` and ran the code using the commands

```
$ gcc static.c -o static
$ ./static
```

we would always get the same output, with 100 points used for `x` and `f`. But what if we wanted to change to 200, or 2000, or some other number of points?

Answer: you would need to change the value of `N` and then recompile. You could save different executables. But this isn't very efficient.

Better answer: what if we could write the application to take the problem size at run time $\Longrightarrow$ dynamic memory allocation!

Dynamic Arrays in C

C allows you to allocate arrays on the fly. Here's how. First the new memory allocation goes like this:

```
double *x, *f;
x = (double *)malloc(N*sizeof(double));
f = (double *)malloc(N*sizeof(double));
< do work with x and f >
free(x); free(f);
```

instead of

```
double x[N], f[N];
```

Dynamic Arrays in C

Let's examine the new code bit-by-bit. First the declaration:

```
double *x, *f;
```

This new notation, which you've seen before in `*argv[]` from HW10, is a pointer. We will talk about it in a future Video.

Now for the two commands,

```
x = (double *)malloc(N*sizeof(double));
f = (double *)malloc(N*sizeof(double));
```

The function `malloc` stands for "memory allocation" and it tells the computer to save a block of memory, associated with `x` or `f`, in the amount of `N*sizeof(double)`.

Dynamic Arrays in C

Let's examine the new code bit-by-bit. First the declaration:

```
double *x, *f;
```

This new notation, which you've seen before in `*argv[]` from HW10, is a pointer. We will talk about it in a future Video.

Now for the two commands,

```
x = (double *)malloc(N*sizeof(double));
f = (double *)malloc(N*sizeof(double));
```

The function `malloc` stands for "memory allocation" and it tells the computer to save a block of memory, associated with `x` or `f`, in the amount of `N*sizeof(double)`.

What about `sizeof()`?

Dynamic Arrays in C

The `sizeof()` function gives information about the amount of memory required to store a given type of variable. For example

variable type	`sizeof()`	fact
`char`	1 byte	—
`int`	4 bytes	$(2^{31} - 1) = 2{,}147{,}483{,}647$
`float`	4 bytes	7 sig. figures
`double`	8 bytes	15 sig. figures

Dynamic Arrays in C

The `sizeof()` function gives information about the amount of memory required to store a given type of variable. For example

variable type	`sizeof()`	fact
`char`	1 byte	—
`int`	4 bytes	$(2^{31} - 1) = 2{,}147{,}483{,}647$
`float`	4 bytes	7 sig. figures
`double`	8 bytes	15 sig. figures

As detailed on this webpage, the 4- and 8-bytes of storage of floats and doubles is split between the mantissa and the exponent.

Dynamic Arrays in C

Notice also the new line

```
free(x); free(f);
```

Dynamic Arrays in C

Notice also the new line

```
free(x); free(f);
```

When you dynamically allocate memory, C requires you to free it, too!

Dynamic Arrays in C

Let's now modify our code `static.c` to accept the problem size at the command line. Call the new code `dynamic.c`.

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

int main (int argc, char *argv[])
{
  int i, N;
  double *x, *f;
  double pi = acos(-1.0);

  /* check command line */
  if (argc != 2) {
    fprintf(stdout,
            "Usage: %s problemsize\n", argv[0]);
    return EXIT_FAILURE;
  }

  /* copy command line argument */
  sscanf(argv[1], "%d", &N);
  /* could also use N = atoi(argv[1]); */

  /* allocate memory */
  x = (double *)malloc(N*sizeof(double));
  f = (double *)malloc(N*sizeof(double));

  /* run through loops */
  for (i = 0; i < N; i++) {
    x[i] = (double)(i) / ((double)(N) - 1.0);
    f[i] = sin(2.0 * pi * x[i]);
  }

  /* free memory */
  free(x); free(f);

  /* all done! */
  return EXIT_SUCCESS;
}
```

Notice the new `#include <math.h>`.

Dynamic Arrays in C

To compile, use this command

```
$ gcc dynamic.c -o dynamic -lm
```

which has the new command option `-lm` to include the math libraries. You must use `-lm` when you use `#include <math.h>`.