Embedded computing for scientific and industrial imaging applications

Lecture 6 - C, data type, function, array, pointer

Outline

- Compiled languages
- Introduction to C syntax
- Declaring variables, loops, booleans
- functions
- Arrays
- Dynamic memory
- Pointer

Compiled vs. interpreted language

Not so much a feature of language syntax as of how language is converted into machine instructions.

Many languages use elements of both.

Interpreter:

- Takes commands one at a time, converts into machine code, and executes. (REPL)
- Allows interactive programming at a shell prompt, as in Python or Matlab.
- Can’t take advantage of optimizing over a entire program — does not know what instructions are coming next.
- Must translate each command while running the code, possibly many times over in a loop.

C history

C (/ˈsiː/, as in the letter c) is a general-purpose, imperative computer programming language

C was originally developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs and used to re-implement the Unix operating system.

C has been standardized by the American National Standards Institute (ANSI) since 1989 (see ANSI C) and subsequently by the International Organization for Standardization (ISO).

ANSI C (C89) -> C99 -> C11

Notes : Visual C++ does not fully support standard C (C99) yet.

C syntax

https://en.wikipedia.org/wiki/C_syntax

Indentation is optional (but highly recommended).

Use file extension `.c`

line is ended with ; (semicolon).
separated with {} (curly brace).

Simple C program

example1.c

```
#include <stdio.h>

int main()
{
    double x, y, z;
    x = 3.0;
    y = 1e-1;
    z = x + y;
    printf("z = %f\n", z);
    return 0;
}
```

Indentation is optional.

First declaration of variables then executable statements
double (double precision floating point number 8byte)

y = 1e-1 means 1×10^{-1}

printf (... %f ...) means print floating point.
printf format string

Compiling and running C

Suppose example1.c contains this program. (or build solution in visual studio)

```
$ cl example1.c
```

compiles and links and creates an executable named `example1.exe`
To run the code after compiling it:

```
$ example1.exe
z = 3.100000
```

Compile-time errors

example1.c

```
#include <stdio.h>

int main()
{
    double x, y, z;
    x = 3.0;
    y = 1e-1;
    zz = x + y;
    printf("z = %f\n", z);
    return 0;
}
```

Introduce an error in the code: (`zz` instead of `z`)

This gives an error when compiling:

```
example1.c(8): error C2065: 'zz' :
undeclared identifier
```

Arrays and loops

```
#include <stdio.h>

#define N 10000

int main()
{
    double x[N], y[N];
    int i;

    for (i = 0; i < N; i++)
    {
        x[i] = 3.0 * i;
    }

    for (i = 0; i < N; i++)
    {
        y[i] = 2.0 * x[i];
    }

    printf("Last y computed: %f\n", y[N - 1]);
    return 0;
}
```

There are two simple ways in C to define constants

- Using `#define` preprocessor to define constants
- Using `const` keyword.

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows

```
type arrayName [ arraySize ];
```

Arrays and loops

```
#include <stdio.h>

#define N 10000

int main()
{
    double x[N], y[N];
    int i;

    for (i = 0; i < N; i++)
    {
        x[i] = 3.0 * i;
    }

    for (i = 0; i < N; i++)
    {
        y[i] = 2.0 * x[i];
    }

    printf("Last y computed: %f\n", y[N - 1]);
    return 0;
}
```

0-based addressing.

`x[i]` means `i+1`'th element of array.

```
for ( init; condition; increment )
{
   statement(s);
}
```

if - else statement

```c
#include <stdio.h>

int main()
{
    int i;
    i = 3;

    if (i <= 2)
    {
        printf("i is less of equal to 2\n");
    }
    else if (i != 5)
    {
        printf("i is greater than 2, not equal to 5\n");
    }
    else
    {
        printf("i is equal to 5\n");
    }
    return 0;
}
```

Booleans

- data type : `bool`
- value : `true`, `false`

Comparisons

- ==, !=, >=, <=, >, <

C operators

functions

For now, assume we have a single file `filename.c` that contains the main function and also any functions needed.

Later we will see how to split into separate files.

Functions take some input arguments and return a single value.

Usage: `y = f(x)` or `z = g(x,y)`

```
return_type function_name( parameter list ) {
   body of the function
}
```

functions - example

```
/* function returning the max between two
numbers */
int max(int num1, int num2) {
   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

Function Declarations

- A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

```
return_type function_name( parameter list );
```

Array operations

There is no array operations in C like MATLAB or FORTRAN.

You should write your own code with for-loop.

Or, Use well defined libraries such as LAPACK (or Intel MKL).

Linear systems in C

There is no equivalent of the Matlab backslash operator for solving a linear system Ax = b (b = A\b)

Must call a library subroutine to solve a system.

Later we will see how to use LAPACK (Intel MKL) for this.

Note: Under the hood, Matlab calls LAPACK too!

Multi-dimensional array storage

SKIP!

Memory management for arrays

Often a program needs to be written to handle arrays whose size is not known until the program is running.

Fortran 77 approaches:

- Allocate arrays large enough for any application,
- Use “work arrays” that are partitioned into pieces.

We will look at some examples from LAPACK since you will probably see this in other software!

The good news: C allows dynamic memory allocation

Memory allocation

```
#include <stdlib.h>

int main()
{
     double* pX;
     int n = 1024;
     pX = malloc(n * sizeof(double));

     //do something

     //clean up
     free(pX);
     return 0;
}
```

Memory allocation

If you might run out of memory, malloc() returns NULL

```
pX = malloc(n * sizeof(double));

if (pX == NULL)
{
    printf("Insufficient memory\n");
    return 0;
}
```

Access violation (segmentation fault)

```
#include <stdlib.h>

int main()
{
     double* pX;
     int n = 1024;
     pX = malloc(n * sizeof(double));

     px[2000] = 1.0;

     //clean up
     free(pX);
     return 0;
}
```

This compiles fine, but running it gives:

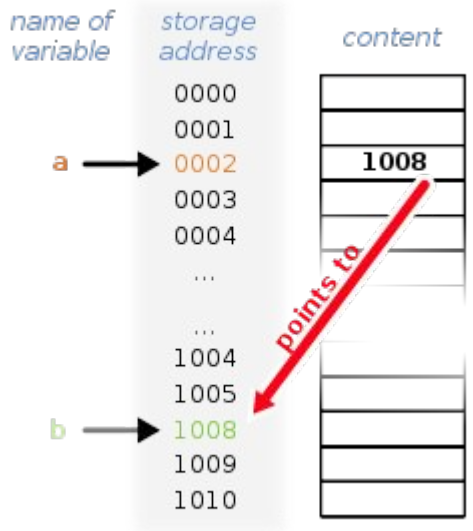
```
Unhandled exception at 0x00953DA6 in
memoryAllocation1.exe: 0xC0000005: Access violation
writing location 0x00F82E58.
```

This means that the program tried to change a value of memory it was not allowed to.

The memory we tried to access might be where the program itself is stored, or something related to another program that’s running.

Pointer

- a programming language object, whose value refers to (or "points to") another value stored elsewhere in the computer memory using its memory address.
- references a location in memory, and obtaining the value stored at that location is known as dereferencing the pointer
- Address unit : 1 byte.



Pointer

```
int* ptr;
```

1. `&` operator - address operator
    a. Get address of variable
2. `*` operator - value at operator
    a. Get value stored at particular address
    b. Declaration of pointer