600.120 Intermediate Programming, Spring 2017*

Misha Kazhdan

**Much of the code in these examples is not commented because it would otherwise not fit on the slides. This is bad coding practice in general and you should not follow my lead on this.*

Outline

- `sizeof` operator
- Static arrays
- Strings
- Printing to and reading from the console

Last time

- Integer types:
  - [unsigned] char: [un]signed character (typically 1 byte)
  - [unsigned] int: [un]signed integer (typically 4 bytes)

- Floating-point types:
  - float: single-precision floating point number (typically 4 bytes)
  - double: double-precision floating point number (typically 8 bytes)

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

sizeof operator

- To determine the size of a type, you can use **sizeof**.

```
#include <stdio.h>
int main(void)
{
        int x = 75;
        printf( "Size of char: %d\n" , sizeof( char ) );
        printf( "Size of int: %d\n" , sizeof( x ) );
        return 0;
}
```

```
>> ./a.out
Size of char: 1
Size of int: 4
>>
```

Static arrays

- Static arrays are declared/accessed using square brackets:

```
#include <stdio.h>
int main(void)
{
        int values[2];
        values[0] = 0;
        values[1] = 130;
        printf( "Array values: %d %d\n" , values[0] , values[1] );
        return 0;
}
```

```
>> ./a.out
Array values: 0 130
>>
```

Static arrays

- Static arrays are declared/accessed using square brackets:
- C/C++ **does not** stop you from accessing values outside the array:

```
#include <stdio.h>
int main(void)
{
        int values[2];
        values[0] = 0;
        values[1] = 130;
        printf( "Array values: %d %d\n" , values[0] , values[2] );
        return 0;
}
```

```
>> ./a.out
Array values: 0 0
>>
```

Static arrays

- Static arrays are declared/accessed using square brackets:
- C/C++ **does not** stop you from accessing values outside the array

```
#include <stdio.h>
int main(void)
{
        int values[2];
        values[0] = 0;
        values[1] = 130;
        printf( "Array values: %d %d\n" , values[0] , values[1024] );
        return 0;
}
```

```
>> ./a.out
Array values: 0 813401299
>>
```

Static arrays

- Static arrays are declared/accessed using square brackets:
- C/C++ **does not** stop you from accessing values outside the array

```
#include <stdio.h>
int main(void)
{
        int x = 100;
        int values[2];
        int y = 100;
        values[0] = 0 ; values[1] = 1 ; values[2] = 2;
        printf( "values = { %d , %d  } , y = %d\n" , values[0] , values[1] , y );
        return 0;
}
```

```
>> ./a.out
values = { 0 , 1 } , y = 2
>>
```

Static arrays

- Static arrays are declared/accessed using square brackets:
- C/C++ **does not** stop you from accessing values outside the array

```
#include <stdio.h>
int main(void)
{
        int x = 100;
        int values[2];
        int y = 100;
        values[0] = 0 ; values[1] = 1 ; values[1000000] = 2;
        printf( "values = { %d , %d  } , y = %d\n" , values[0] , values[1] , y );
        return 0;
}
```

```
>> ./a.out
Segmentation fault (core dumped)
>>
```

Static arrays

- Static arrays are declared/accessed using square brackets:
- C/C++ **does not** stop you from accessing values outside the array:
- You can declare and assign array values at the same time
  - The array size is automatically determined from the assignment

```
#include <stdio.h>
int main(void)
{
        int values[] = { 0 , 130 };
        printf( "Array values: %d %d\n" , values[0] , values[1] );
        return 0;
}
```

```
>> ./a.out
Array values: 0 130
>>
```

Static arrays

- You can determine the size of a static array using the **sizeof** operator

```
#include <stdio.h>
int main(void)
{
        int values[] = { 0 , 130 };
        printf( "Array size: %d\n" , sizeof( values ) );
        return 0;
}
```

```
>> ./a.out
Array size: 8
>>
```

Q: Why does the array have size 8 if it only has two entries?

Strings

- Strings are arrays of <u>null-terminated</u> characters
  - The null termination is required to indicate where the string ends
    - The character '\0' has value 0, so either is fine

```
#include <stdio.h>
int main(void)
{
        char str[] = { 'h' , 'e' , 'l' , 'l' , 'o' , '\0' };
        printf( "str: %s\n" , str );
        return 0;
}
```

```
>> ./a.out
str: hello
>>
```

Strings

- Strings are arrays of <u>null-terminated</u> characters
  - The null termination is required to indicate where the string ends
    - The character '\0' has value 0, so either is fine
    - The character '\n' is a new-line
    - The character '\t' is a tab
    - The character '\"' is a quote
    - etc.

```
#include <stdio.h>
int main(void)
{
        char str[] = { 'h' , 'e' , 'l' , 'l' , 'o' , '\0' };
        printf( "str: %s\n" , str );
        return 0;
}
```

```
>> ./a.out
str: hello
>>
```

Strings

- Strings are arrays of <u>null-terminated</u> characters
  - The null termination is required to indicate where the string ends
- Can use double-quotes to assign the string value

```
#include <stdio.h>
int main(void)
{
        char str[] = "hello";
        printf( "str: %s\n" , str );
        return 0;
}
```

```
>> ./a.out
str: hello
>>
```

Strings

- Strings are arrays of <u>null-terminated</u> characters
  - The null termination is required to indicate where the string ends
- Can use double-quotes to assign the string value
  - Multiple quoted strings are merged into one long string
    - Makes it possible to split text across multiple lines

```
#include <stdio.h>
int main(void)
{
        char str[] = "hel"
                     "lo";
        printf( "str: %s\n" , str );
        return 0;
}
```

```
>> ./a.out
str: hello
>>
```

String functions (declared in `string.h`)

- **strlen**: Get the length of a string

```
#include <stdio.h>
#include <string.h>
int main( void )
{
        char str[] = "hello";
        printf( "string length : %d\n" , strlen( str ) );
        return 0;
}
```

```
>> ./a.out
string length: 5
>>
```

String functions (declared in `string.h`)

- **strlen**: Get the length of a string

```
#include <stdio.h>
#include <string.h>
int main( void )
{
        char str[] = "hello";
        printf( "string length / size: %d %d\n" , strlen( str ) , sizeof( str ) );
        return 0;
}
```

```
>> ./a.out
string length / size: 5 6
>>
```

Q: Why are the length and size different?

String functions (declared in `string.h`)

- `strcpy`: Copy the contents of one string into the other
  - The target must be large enough to store the source (and its null-terminator)

```
#include <stdio.h>
#include <string.h>
int main(void)
{
        char source[] = "hello";
        char target[6];
        strcpy( target , source );
        printf( "string: %s\n" , target );
        return 0;
}
```

```
>> ./a.out
string: hello
>>
```

String functions (declared in `string.h`)

- **strcmp**: Compare two strings
  - returns > 0: If the first string comes before the second
  - returns < 0: If the second string comes before the first
  - returns 0: if the strings are equal

```
#include <stdio.h>
#include <string.h>
int main(void)
{
        char str1[] = "hello";
        char str2[] = "goodbye";
        printf( "compare( %s , %s ) = %d\n" , str1 , str2 , strcmp( str1 , str2 ) );
        return 0;
}
```

```
>> ./a.out
compare( hello , goodbye ) = 1
>>
```

String functions (declared in `string.h`)

- `strtok`: Tokenizes a string
- `strcat`: Concatenates two strings
- and much much more

http://www.cplusplus.com/reference/cstring/

String functions (declared in `stdlib.h`)

- `atoi`: converts a string into a an integer
- `atof`: converts a string into a (double-precision) floating point value

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
        char str[] = "120";
        int i = atoi( str );
        double d = atof( str );
        printf( "%s -> %d : %f\n" , str, i , d );
        return 0;
}
```

```
>> ./a.out
120 -> 120 : 120.000000
>>
```

Printing to the console

- void putchar(char c):
  Outputs a single character to the command prompt (standard out)

```
#include <stdio.h>
#include <string.h>
int main(void)
{
        char str[] = "hello";
        for( int i=0 ; str[i]!=0 ; i++ ) putchar( str[i] );
        putchar( '\n' );
        return 0;
}
```

```
>> ./a.out
hello
>>
```

Printing to the console

- **int printf(const char format_str[] , …):**
  Prints a (formatted) string to the command prompt (standard out)

```
#include <stdio.h>
#include <string.h>
int main(void)
{
        char str[] = "hello";
        printf( "%s\n" , str );
        return 0;
}
```

```
>> ./a.out
hello
>>
```

Printing to the console

- Formally, **printf** is a <u>variadic</u> function taking a string, followed by an arbitrary number of arguments:

```
int printf( const char format_str[] , ... );
```

Printing to the console

- Formally, `printf` is a <u>variadic</u> function taking a string, followed by an arbitrary number of arguments:

```
int printf( const char format_str[] , ... );
```

- In practice, it
  - writes the characters of the first string to the command prompt
  - if it encounters a special character it writes out the next argument.
    - `%d`: the next argument is an integer
    - `%f`: the next argument is floating point number
    - `%c`: the next argument is a character
    - `%s`: the next argument is a (null-terminated) string
    - etc.

Printing to the console

- Formally, **printf** is a <u>variadic</u> funct
  arbitrary number of arguments:

  int printf(const ch

- In practice, it
  - writes the characters of the first strin
  - if it encounters a special character it writes out the
    - %d: the next argument is an integer
    - %f: the next argument is floating point number

```
#include <stdio.h>
int main(void)
{
        char str[] = "C3";
        char c = 'P';
        int i = 0;
        printf( "%s%c%d\n" , str , c , i );
        return 0;
}
```

```
>> ./a.out
C3P0
>>
```

Make sure that the number of arguments matches the number of format tags

- The compiler will throw a warning, but will still generate executable code.

Printing to the console

- You can provide further flags as to how things should be printed:
  - %<j>d: At least <j> spaces should be used to print the number

```
#include <stdio.h>
int main(void)
{
        int x = 123;
        printf( "x=%2d : x=%4d\n" , x , x );
        return 0;
}
```

```
>> ./a.out
x=123 : x= 123
>>
```

Printing to the console

- You can provide further flags as to how things should be printed:
  - %<j>.<k>f: At least <j> spaces should be used to print the number and <k> decimals of precision should be used

```
#include <stdio.h>
int main(void)
{
        float x = 1.484;
        printf( "x=%4.1f\n" , x );
        return 0;
}
```

```
>> ./a.out
x= 1.5
>>
```

Note: numbers will be rounded if the precision isn't large enough

Printing to the console

- You can provide further flags as to how things should be printed:
  - and much much more

http://www.cplusplus.com/reference/cstdio/printf/

Printing to the console

Q: What does this code output?

```
#include <stdio.h>
int main(void)
{
        char str[] = "good bye";
        printf( "str=\"%s\"\n" , str );
        str[ strlen( "good" ) ] = 0;
        printf( "str=\"%s\"\n" , str );
        return 0;
}
```

```
>> ./a.out
str="good bye"
str="good"
>>
```

Printing to the console

Q: What does this code output?

Q: How about this?

```
#include <stdio.h>
int main(void)
{
        char str[] = "good bye";
        printf( "%d %d\n" , strlen( str) , sizeof( str ) );
        str[ strlen( "good" ) ] = 0;
        printf( "%d %d\n" , strlen( str) , sizeof( str ) );
        return 0;
}
```

```
>> ./a.out
8 9
4 9
>>
```

Reading from the console

- getchar:
  Reads the next character from the command prompt (standard in) buffer
  - It waits until [ENTER] is hit before returning anything
  - Calling it a second time, will return the second character read
  - etc.

```
#include <stdio.h>
int main(void)
{
        char c;
        while( 1 )
        {
                c = getchar();
                putchar( c );
                if( c=='\n' ) break;
        }
        return 0;
}
```

Putting it together

```
#include <stdio.h>
int main( void )
{
        const int base = 32;
        const float factor = 5.0 / 9.0;
        int fahrenheit = 75;
        float celsius = factor * (fahrenhe
        printf( "%d -> %0.2f\n" , farenhe
        return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
{
        const int base = 32;
        const float factor = 5.0 / 9.0;
        while( 1 )
        {
                char str[16];
                int idx = 0;
                printf( "Enter a temperature: " );
                while( ( str[idx]=getchar() )!='\n' ) idx++;
                if( !idx ) break;
                str[idx] = 0;
                int farenheit = atoi( str );
                float celcius = factor * (farenheit - base);
                printf( "%d -> %0.2f\n" , farenheit , celcius );
        }
        return 0;
}
```

Q: What is wrong with this code?

In-Class Exercises

- On Piazza, find Resources section, then click Resources tab
- Scroll down to section for this course section
- Find link for Exercise 2-1 and follow it
- Follow the instructions; raise your hand if you get stuck
- Make sure you check in with a course staff member sometime during this session