CMPUT201W20B2 Week 3

Abram Hindle

March 10, 2020

Contents

1 Week3

1.1 Copyright Statement

1.1.1 License

1.1.2 Hazel Code is licensed under AGPL3.0+

Hazel's code is also found here `https://github.com/hazelybell/examples/tree/C-2020-01`

Hazel code is licensed: The example code is licensed under the AGPL3+ license, unless otherwise noted.

1.2 Init ORG-MODE

;; I need this for org-mode to work well

(require 'ob-sh) ;(require 'ob-shell) (org-babel-do-load-languages 'org-babel-load-languages '((sh . t))) (org-babel-do-load-languages 'org-babel-load-languages '((C . t))) (org-babel-do-load-languages 'org-babel-load-languages '((python . t))) (setq org-src-fontify-natively t)

1.2.1 Org export

```
(org-html-export-to-html)
(org-latex-export-to-pdf)
(org-ascii-export-to-ascii)
```

1.3 Org Template

Copy and paste this to demo C

```
#include <stdio.h>

int main(int argc, char**argv) {
    return 0;
}
```

1.4 Remember how to compile?

gcc -std=c99 -Wall -pedantic -Werror -o programname programname.c

1.5 Types!

1.5.1 int!

In C ints are often 32-bit integers. They can have a sign.

```
#include <stdio.h>
#include <limits.h>
int main() {
    int an_int = 6;
    printf("size_of(an_int) == %ld\n", sizeof(an_int));
    int max_int = INT_MAX;
    printf("max int == %11d\n", max_int);
    int min_int = INT_MIN;
    printf("min int == %11d\n", min_int);
    printf("an_int  == %11d \t== 0x%08x\n", an_int,  an_int);
    printf("min_int == %11d \t== 0x%08x\n", min_int, min_int);
    printf("max_int == %11d \t== 0x%08x\n", max_int, max_int);
    printf("     -1 == %11d \t== 0x%08x\n", -1,-1);
    printf("      1 == %11d \t== 0x%08x\n", 1,1;)
    printf("      0 == %11d \t== 0x%08x\n", 0,0);

    return 0;
}
```

1.5.2 Signed Datatypes [hazel]

```
./signed.c

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h> // <-- _MIN _MAX
#include <float.h>
#include <stdint.h>

int main() {
    signed char scmax = SCHAR_MAX;
    signed char scmin = SCHAR_MIN;
    printf("char: %zu bytes %zu bits %hhd to %hhd\n",
           sizeof(char),
           sizeof(char) * 8,
           scmin,
           scmax
    );
    short smax = SHRT_MAX;
    short smin = SHRT_MIN;
    printf("short: %zu bytes %zu bits %hd to %hd\n",
           sizeof(short),
           sizeof(short) * 8,
           smin,
           smax
    );
    int imax = INT_MAX;
    int imin = INT_MIN;
    printf("int: %zu bytes %zu bits %d to %d\n",
           sizeof(int),
           sizeof(int) * 8,
           imin,
           imax
    );
    long lmax = LONG_MAX;
    long lmin = LONG_MIN;
    printf("long: %zu bytes %zu bits %ld to  %ld\n",
           sizeof(long),
           sizeof(long) * 8,
           lmin,
           lmax
    );
    long long llmax = LLONG_MAX;
    long long llmin = LLONG_MIN;
    printf("long long: %zu bytes %zu bits %lld to %lld\n",
           sizeof(long long),
           sizeof(long long) * 8,
           llmin,
           llmax
    );
    float fmax = FLT_MAX;
    float fmin = FLT_MIN;
    printf("float: %zu bytes %zu bits %e to %e\n",
           sizeof(float),
           sizeof(float) * 8,
           fmin,
           fmax
    );
    double dmax = DBL_MAX;
    double dmin = DBL_MIN;
    printf("double: %zu bytes %zu bits max %e to %e\n",
           sizeof(double),
           sizeof(double) * 8,
           dmin,
            dmax
    );
    long double ldmax = LDBL_MAX;
    long double ldmin = LDBL_MIN;
    printf("long double: %zu bytes %zu bits  %Le to %Le\n",
            sizeof(long double),
            sizeof(long double) * 8,
            ldmin,
            ldmax
    );
    return 0;
}
```

```
char: 1 bytes 8 bits -128 to 127
short: 2 bytes 16 bits -32768 to 32767
int: 4 bytes 32 bits -2147483648 to 2147483647
long: 8 bytes 64 bits -9223372036854775808 to  9223372036854775807
long long: 8 bytes 64 bits -9223372036854775808 to 9223372036854775807
float: 4 bytes 32 bits 1.175494e-38 to 3.402823e+38
double: 8 bytes 64 bits max 2.225074e-308 to 1.797693e+308
long double: 16 bytes 128 bits  3.362103e-4932 to 1.189731e+4932
```

1.5.3 unsigned ints!

You can only non-negative integers if you want

```
#include <stdio.h>
#include <limits.h>
int main() {
    unsigned int an_int = 6;
    printf("size_of(an_int) == %ld\n", sizeof(an_int));
    unsigned int max_int = UINT_MAX;
    printf("max int == %11u\n", max_int);
    unsigned int min_int = 0;
    printf("min int == %11u\n", min_int);
    printf("an_int  == %11u \t== 0x%08x\n", an_int,  an_int);
    printf("min_int == %11u \t== 0x%08x\n", min_int, min_int);
    printf("max_int == %11u \t== 0x%08x\n", max_int, max_int);
    printf("     -1 == %11u \t== 0x%08x\n", -1,-1);
    printf("      1 == %11u \t== 0x%08x\n", 1,1);
    printf("      0 == %11u \t== 0x%08x\n", 0,0);
    return 0;
}

size_of(an_int) == 4
max int ==  4294967295
min int ==           0
an_int  ==           6  == 0x00000006
min_int ==           0  == 0x00000000
max_int ==  4294967295  == 0xffffffff
     -1 ==  4294967295  == 0xffffffff
      1 ==           1  == 0x00000001
      0 ==           0  == 0x00000000
```

1.5.4 Ints [hazel]

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

/* For more info:
 * https://en.cppreference.com/w/c/types/integer
 */

int main() {
    printf("int8_t: %zu bytes %zu bits\n",
           sizeof(int8_t),
           sizeof(int8_t) * 8
    );
    printf("int16_t: %zu bytes %zu bits\n",
           sizeof(int16_t),
           sizeof(int16_t) * 8
    );
    printf("int32_t: %zu bytes %zu bits\n",
           sizeof(int32_t),
           sizeof(int32_t) * 8
    );
    printf("int64_t: %zu bytes %zu bits\n",
           sizeof(int64_t),
           sizeof(int64_t) * 8
    );
    printf("intmax_t: %zu bytes %zu bits\n",
           sizeof(intmax_t),
           sizeof(intmax_t) * 8
    );
    printf("int_fast8_t: %zu bytes %zu bits\n",
           sizeof(int_fast8_t),
           sizeof(int_fast8_t) * 8
    );
    printf("int_fast16_t: %zu bytes %zu bits\n",
           sizeof(int_fast16_t),
           sizeof(int_fast16_t) * 8
    );
    return 0;
}
```

```
int8_t: 1 bytes 8 bits
int16_t: 2 bytes 16 bits
int32_t: 4 bytes 32 bits
int64_t: 8 bytes 64 bits
intmax_t: 8 bytes 64 bits
int_fast8_t: 1 bytes 8 bits
int_fast16_t: 8 bytes 64 bits
```

1.5.5 Unsigned int [hazel]

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include <stdint.h>


int main() {
    unsigned char c = UCHAR_MAX;
    printf("char: %zu bytes %zu bits max %hhu\n",
           sizeof(c),
           sizeof(c) * 8,
           c
    );
    unsigned short s = USHRT_MAX;
    printf("short: %zu bytes %zu bits max %hu\n",
           sizeof(unsigned short),
           sizeof(unsigned short) * 8,
           s
    );
    unsigned int i = UINT_MAX;
    printf("int: %zu bytes %zu bits max %u\n",
           sizeof(i),
           sizeof(i) * 8,
           i
    );
    unsigned long l = ULONG_MAX;
    printf("ulong: %zu bytes %zu bits max %lu\n",
           sizeof(l),
           sizeof(l) * 8,
           l
    );
    unsigned long long ll = ULLONG_MAX;
    printf("long long: %zu bytes %zu bits max %llu\n",
           sizeof(ll),
           sizeof(long long) * 8,
           ll
    );
    size_t z = SIZE_MAX;
    printf("size_t: %zu bytes %zu bits max %zu\n",
           sizeof(z),
           sizeof(z) * 8,
           z
    );
    return 0;
}
```

```
char: 1 bytes 8 bits max 255
short: 2 bytes 16 bits max 65535
int: 4 bytes 32 bits max 4294967295
ulong: 8 bytes 64 bits max 18446744073709551615
long long: 8 bytes 64 bits max 18446744073709551615
size_t: 8 bytes 64 bits max 18446744073709551615
```

1.5.6 Characters and Wrapping

unsigned char Characters have a limited range of 0-255.

```
#include <stdio.h>
int main() {
    unsigned char my_char=0;
    int i = 0;
    for (i = 0 ; i < 512; i++) {
        my_char = i;
        // %hhu is hex signed or unsigned char, u for unsigned int
        printf("my_char=%hhu[%c]",
               my_char,
               (my_char > 127 || my_char < 32)?' ':my_char
        );
        printf("%c", ( (i+1) % 8 == 0 )?'\n':'\t');
    }
    printf("We did iterate over i=%d iterations\n", i);
}
```

```
my_char=0[ ] my_char=1[ ] my_char=2[ ] my_char=3[ ] my_char=4[ ]
my_char=5[ ] my_char=6[ ] my_char=7[ ]
my_char=8[ ] my_char=9[ ] my_char=10[ ] my_char=11[ ] my_char=12[ ]
my_char=13[ ] my_char=14[ ] my_char=15[ ]
my_char=16[ ] my_char=17[ ] my_char=18[ ] my_char=19[ ] my_char=20[ ]
my_char=21[ ] my_char=22[ ] my_char=23[ ]
my_char=24[ ] my_char=25[ ] my_char=26[ ] my_char=27[ ] my_char=28[ ]
my_char=29[ ] my_char=30[ ] my_char=31[ ]
my_char=32[ ] my_char=33[!] my_char=34["] my_char=35[#] my_char=36[$]
my_char=37[%] my_char=38[&] my_char=39[']
my_char=40[(] my_char=41[)] my_char=42[*] my_char=43[+] my_char=44[,]
my_char=45[-] my_char=46[.] my_char=47[/]
my_char=48[0] my_char=49[1] my_char=50[2] my_char=51[3] my_char=52[4]
my_char=53[5] my_char=54[6] my_char=55[7]
my_char=56[8] my_char=57[9] my_char=58[:] my_char=59[;] my_char=60[<]
my_char=61[=] my_char=62[>] my_char=63[?]
my_char=64[@] my_char=65[A] my_char=66[B] my_char=67[C] my_char=68[D]
my_char=69[E] my_char=70[F] my_char=71[G]
my_char=72[H] my_char=73[I] my_char=74[J] my_char=75[K] my_char=76[L]
my_char=77[M] my_char=78[N] my_char=79[O]
```

```
my_char=80[P] my_char=81[Q] my_char=82[R] my_char=83[S] my_char=84[T]
my_char=85[U] my_char=86[V] my_char=87[W]
my_char=88[X] my_char=89[Y] my_char=90[Z] my_char=91[[] my_char=92[\]
my_char=93[]] my_char=94[^] my_char=95[_]
my_char=96[`] my_char=97[a] my_char=98[b] my_char=99[c] my_char=100[d]
my_char=101[e] my_char=102[f] my_char=103[g]
my_char=104[h] my_char=105[i] my_char=106[j] my_char=107[k] my_char=108[l]
my_char=109[m] my_char=110[n] my_char=111[o]
my_char=112[p] my_char=113[q] my_char=114[r] my_char=115[s] my_char=116[t]
my_char=117[u] my_char=118[v] my_char=119[w]
my_char=120[x] my_char=121[y] my_char=122[z] my_char=123[{] my_char=124[|]
my_char=125[}] my_char=126[~] my_char=127[]
my_char=128[ ] my_char=129[ ] my_char=130[ ] my_char=131[ ] my_char=132[ ]
my_char=133[ ] my_char=134[ ] my_char=135[ ]
my_char=136[ ] my_char=137[ ] my_char=138[ ] my_char=139[ ] my_char=140[ ]
my_char=141[ ] my_char=142[ ] my_char=143[ ]
my_char=144[ ] my_char=145[ ] my_char=146[ ] my_char=147[ ] my_char=148[ ]
my_char=149[ ] my_char=150[ ] my_char=151[ ]
my_char=152[ ] my_char=153[ ] my_char=154[ ] my_char=155[ ] my_char=156[ ]
my_char=157[ ] my_char=158[ ] my_char=159[ ]
my_char=160[ ] my_char=161[ ] my_char=162[ ] my_char=163[ ] my_char=164[ ]
my_char=165[ ] my_char=166[ ] my_char=167[ ]
my_char=168[ ] my_char=169[ ] my_char=170[ ] my_char=171[ ] my_char=172[ ]
my_char=173[ ] my_char=174[ ] my_char=175[ ]
my_char=176[ ] my_char=177[ ] my_char=178[ ] my_char=179[ ] my_char=180[ ]
my_char=181[ ] my_char=182[ ] my_char=183[ ]
my_char=184[ ] my_char=185[ ] my_char=186[ ] my_char=187[ ] my_char=188[ ]
my_char=189[ ] my_char=190[ ] my_char=191[ ]
my_char=192[ ] my_char=193[ ] my_char=194[ ] my_char=195[ ] my_char=196[ ]
my_char=197[ ] my_char=198[ ] my_char=199[ ]
my_char=200[ ] my_char=201[ ] my_char=202[ ] my_char=203[ ] my_char=204[ ]
my_char=205[ ] my_char=206[ ] my_char=207[ ]
my_char=208[ ] my_char=209[ ] my_char=210[ ] my_char=211[ ] my_char=212[ ]
my_char=213[ ] my_char=214[ ] my_char=215[ ]
my_char=216[ ] my_char=217[ ] my_char=218[ ] my_char=219[ ] my_char=220[ ]
my_char=221[ ] my_char=222[ ] my_char=223[ ]
my_char=224[ ] my_char=225[ ] my_char=226[ ] my_char=227[ ] my_char=228[ ]
my_char=229[ ] my_char=230[ ] my_char=231[ ]
my_char=232[ ] my_char=233[ ] my_char=234[ ] my_char=235[ ] my_char=236[ ]
my_char=237[ ] my_char=238[ ] my_char=239[ ]
```

```
my_char=240[ ] my_char=241[ ] my_char=242[ ] my_char=243[ ] my_char=244[ ]
my_char=245[ ] my_char=246[ ] my_char=247[ ]
my_char=248[ ] my_char=249[ ] my_char=250[ ] my_char=251[ ] my_char=252[ ]
my_char=253[ ] my_char=254[ ] my_char=255[ ]
my_char=0[ ] my_char=1[ ] my_char=2[ ] my_char=3[ ] my_char=4[ ]
my_char=5[ ] my_char=6[ ] my_char=7[ ]
my_char=8[ ] my_char=9[ ] my_char=10[ ] my_char=11[ ] my_char=12[ ]
my_char=13[ ] my_char=14[ ] my_char=15[ ]
my_char=16[ ] my_char=17[ ] my_char=18[ ] my_char=19[ ] my_char=20[ ]
my_char=21[ ] my_char=22[ ] my_char=23[ ]
my_char=24[ ] my_char=25[ ] my_char=26[ ] my_char=27[ ] my_char=28[ ]
my_char=29[ ] my_char=30[ ] my_char=31[ ]
my_char=32[ ] my_char=33[!] my_char=34["] my_char=35[#] my_char=36[$]
my_char=37[%] my_char=38[&] my_char=39[']
my_char=40[(] my_char=41[)] my_char=42[*] my_char=43[+] my_char=44[,]
my_char=45[-] my_char=46[.] my_char=47[/]
my_char=48[0] my_char=49[1] my_char=50[2] my_char=51[3] my_char=52[4]
my_char=53[5] my_char=54[6] my_char=55[7]
my_char=56[8] my_char=57[9] my_char=58[:] my_char=59[;] my_char=60[<]
my_char=61[=] my_char=62[>] my_char=63[?]
my_char=64[@] my_char=65[A] my_char=66[B] my_char=67[C] my_char=68[D]
my_char=69[E] my_char=70[F] my_char=71[G]
my_char=72[H] my_char=73[I] my_char=74[J] my_char=75[K] my_char=76[L]
my_char=77[M] my_char=78[N] my_char=79[O]
my_char=80[P] my_char=81[Q] my_char=82[R] my_char=83[S] my_char=84[T]
my_char=85[U] my_char=86[V] my_char=87[W]
my_char=88[X] my_char=89[Y] my_char=90[Z] my_char=91[[] my_char=92[\]
my_char=93[]] my_char=94[^] my_char=95[_]
my_char=96[`] my_char=97[a] my_char=98[b] my_char=99[c] my_char=100[d]
my_char=101[e] my_char=102[f] my_char=103[g]
my_char=104[h] my_char=105[i] my_char=106[j] my_char=107[k] my_char=108[l]
my_char=109[m] my_char=110[n] my_char=111[o]
my_char=112[p] my_char=113[q] my_char=114[r] my_char=115[s] my_char=116[t]
my_char=117[u] my_char=118[v] my_char=119[w]
my_char=120[x] my_char=121[y] my_char=122[z] my_char=123[{] my_char=124[|]
my_char=125[}] my_char=126[~] my_char=127[]
my_char=128[ ] my_char=129[ ] my_char=130[ ] my_char=131[ ] my_char=132[ ]
my_char=133[ ] my_char=134[ ] my_char=135[ ]
my_char=136[ ] my_char=137[ ] my_char=138[ ] my_char=139[ ] my_char=140[ ]
my_char=141[ ] my_char=142[ ] my_char=143[ ]
```

```
my_char=144[ ] my_char=145[ ] my_char=146[ ] my_char=147[ ] my_char=148[ ]
my_char=149[ ] my_char=150[ ] my_char=151[ ]
my_char=152[ ] my_char=153[ ] my_char=154[ ] my_char=155[ ] my_char=156[ ]
my_char=157[ ] my_char=158[ ] my_char=159[ ]
my_char=160[ ] my_char=161[ ] my_char=162[ ] my_char=163[ ] my_char=164[ ]
my_char=165[ ] my_char=166[ ] my_char=167[ ]
my_char=168[ ] my_char=169[ ] my_char=170[ ] my_char=171[ ] my_char=172[ ]
my_char=173[ ] my_char=174[ ] my_char=175[ ]
my_char=176[ ] my_char=177[ ] my_char=178[ ] my_char=179[ ] my_char=180[ ]
my_char=181[ ] my_char=182[ ] my_char=183[ ]
my_char=184[ ] my_char=185[ ] my_char=186[ ] my_char=187[ ] my_char=188[ ]
my_char=189[ ] my_char=190[ ] my_char=191[ ]
my_char=192[ ] my_char=193[ ] my_char=194[ ] my_char=195[ ] my_char=196[ ]
my_char=197[ ] my_char=198[ ] my_char=199[ ]
my_char=200[ ] my_char=201[ ] my_char=202[ ] my_char=203[ ] my_char=204[ ]
my_char=205[ ] my_char=206[ ] my_char=207[ ]
my_char=208[ ] my_char=209[ ] my_char=210[ ] my_char=211[ ] my_char=212[ ]
my_char=213[ ] my_char=214[ ] my_char=215[ ]
my_char=216[ ] my_char=217[ ] my_char=218[ ] my_char=219[ ] my_char=220[ ]
my_char=221[ ] my_char=222[ ] my_char=223[ ]
my_char=224[ ] my_char=225[ ] my_char=226[ ] my_char=227[ ] my_char=228[ ]
my_char=229[ ] my_char=230[ ] my_char=231[ ]
my_char=232[ ] my_char=233[ ] my_char=234[ ] my_char=235[ ] my_char=236[ ]
my_char=237[ ] my_char=238[ ] my_char=239[ ]
my_char=240[ ] my_char=241[ ] my_char=242[ ] my_char=243[ ] my_char=244[ ]
my_char=245[ ] my_char=246[ ] my_char=247[ ]
my_char=248[ ] my_char=249[ ] my_char=250[ ] my_char=251[ ] my_char=252[ ]
my_char=253[ ] my_char=254[ ] my_char=255[ ]
We did iterate over i=512 iterations
```

```c
#include <stdio.h>

int main() {
    unsigned short x = 1;
    while(x!=0) {
        x++;
    }
    printf("x=%u\n",x);
}
```

```
x=0
```

Here's an example from Hazel where we get a negative character and it wraps as well.

```
#include <stdio.h>

int main() {
    unsigned char my_char = 0;
    printf("my_char=%hhu\n", my_char);
    my_char = -10;
    printf("my_char=%hhu\n", my_char);
}
```

```
my_char=0
my_char=246
```

1.5.7 Chars Min Max [hazel]

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h> // <--- _MIN/_MAX
#include <float.h>
#include <stdint.h>

int main() {
    signed char scmax = SCHAR_MAX;
    signed char scmin = SCHAR_MIN;
    printf("signed char: %hhd to %hhd\n",
           scmin,
           scmax
    );
    unsigned char ucmax = UCHAR_MAX;
    unsigned char ucmin = 0;
    printf("UNsigned char: %hhu to %hhu\n",
           ucmin,
           ucmax
    );
    char cmax = CHAR_MAX;
    char cmin = CHAR_MIN;
    printf("plain char: %hhd to %hhd\n",
           cmin,
            cmax
    );
    if (cmin == 0) {
        printf("By default, char is UNsigned!\n");
    } else if (cmin < 0) {
        printf("By default, char is signed!\n");
    } else {
        printf("Error!\n");
        abort();
    }
    return 0;
}
```

```
signed char: -128 to 127
UNsigned char: 0 to 255
plain char: -128 to 127
By default, char is signed!
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <float.h>
#include <stdint.h>


int main(int argc, char ** argv) {
    // (type you want) variable
    printf("%hhd -> %hd\n",
            CHAR_MAX,
            (short) CHAR_MAX);
    printf("%hd -> %hhd\n",
            SHRT_MAX,
            (char) SHRT_MAX);
    short s = 127;
    printf("%hd -> %hhd\n",
            s,
            (char) s);
    s = 128;
    printf("%hd -> %hhd\n",
            s,
            (char) s);
    if (s > CHAR_MAX || s < CHAR_MIN) {
        printf("Error!\n");
    }
    // why?
    printf("0x%hx -> 0x%hhx\n",
            SHRT_MAX,
            (char) SHRT_MAX
    );
    return 0;
}
```

```
127 -> 127
32767 -> -1
127 -> 127
128 -> -128
Error!
0x7fff -> 0xff
```

1.5.8 Reading and Writing Characters

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char ** argv) {
    putchar('H');
    putchar('i');
    putchar('!');
    putchar('\n');
    return 0;
}
```

```
Hi!
```

./getchar-example.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char ** argv) {
  char c = '!';
  int count = 0;
  do {
    c = getchar();
    printf( "%d-", count++);
    putchar(c);
    putchar(' ');
  } while( c != '!' ); // Do while is while loop in reverse!
  return 0;
}
```

Compile it.

```
gcc -std=c99 -Wall -pedantic -Werror -o getchar-example getchar-example.c
```

Run it.

```
echo "Cool bears are up north!" | ./getchar-example
```

```
0-C 1-o 2-o 3-l 4-  5-b 6-e 7-a 8-r 9-s 10-  11-a 12-r 13-e 14-  15-u 16-p 17-  18-n 19
```

1.5.9 Floating Point Numbers!

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

int main() {
    float f1 = 64;
    float f2 = 6.4e1;
    float f3 = 64e0;
    float f4 = 640e-1;
    printf("sizeof(float)=%lu\n",sizeof(float));
    printf("%f %f %f %f\n", f1, f2, f3, f4);
    return 0;
}
```

```
sizeof(float)=4
64.000000 64.000000 64.000000 64.000000
```

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

```
int main() {
    float f1 = 76.1;
    float f2 = 7.61e1;
    float f3 = 76.1e0;
    float f4 = 761e-1;
    printf("sizeof(float)=%lu\n", sizeof(float));
    printf("%f %f %f %f\n", f1, f2, f3, f4);
    return 0;
}
```

```
sizeof(float)=4
76.099998 76.099998 76.099998 76.099998
```

Uh oh. It can't actually represent 76.1.
What if we use bigger numbers?

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

int main() {
    double f1 = 76.1;
    double f2 = 7.61e1;
    double f3 = 76.1e0;
    double f4 = 761e-1;
    printf("sizeof(float)=%lu\n", sizeof(double));
    printf("%lf %lf %lf %lf\n", f1, f2, f3, f4);
    return 0;
}
```

```
sizeof(float)=8
76.100000 76.100000 76.100000 76.100000
```

OK cool. Does that work for all numbers? No. No it does not.

1. Floating Point Operations

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

   int main() {
       float  f1 = 61 / 3;
   ```

```
    int    i1 = 61 / 3;
    double d1 = 61 / 3;
    printf("%f %d %lf\n", f1,i1,d1);
    printf("61 %% 3 = %d\n", 61 % 3); //look how to print %
    printf("(61 - 1) / 3 == 61/3 == %d\n", ((61-1)/3) == (61/3));

    // force floating division
    float  f2 = 61.0 / 3;
    int    i2 = 61.0 / 3;
    double d2 = 61.0 / 3;
    printf("%f %d %lf\n", f2,i2,d2);

    // force floating division
    float  f3 = 61 / (float)3;
    int    i3 = 61 / (float)3;
    double d3 = 61 / (double)3;
    printf("%f %d %lf\n", f3,i3,d3);
    return 0;
}
```

```
20.000000 20 20.000000
61 % 3 = 1
(61 - 1) / 3 == 61/3 == 1
20.333334 20 20.333333
20.333334 20 20.333333
```

1.5.10 Type Definitions

1. #define

   #define SYMBOL value

   defines a macro in C using the C Preprocessor. So all instances of SYMBOL as identifiers (NOT WITHIN STRINGS) will be replaced by the string value

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

   // please no don't do this, example only
   // change to value or notvalue
   ```

```
#define SYMBOL notvalue
// #define SYMBOL value

int main() {
    int value = 10;
    int notvalue = 6;
    printf("%d\n", SYMBOL);
    printf("(SYMBOL == value) %d (SYMBOL == notvalue) %d",
           SYMBOL == value,
           SYMBOL == notvalue
    );
}
```

```
6
(SYMBOL == value) 0 (SYMBOL == notvalue) 1
```

We can use #define to define types too

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> // get some infinity

// please no don't do this, example only
// change to value or notvalue

// #define SYMBOL notvalue
#define HONKIN long double

int main() {
    HONKIN value = 1e300;
    printf("%Lf\n", value);
    printf("%Le\n", value);
    value = INFINITY;
    printf("%Lf\n", value);
    printf("%Le\n", value);
    value = -INFINITY;
    printf("%Lf\n", value);
    printf("%Le\n", value);
    printf("HONKIN SIZE %lu\n", sizeof(value));
}

1000000000000000052504760255204420248704468581108159154915854115511802457988908195
1.000000e+300
inf
inf
-inf
-inf
HONKIN SIZE 16
```

1.5.11 Typedefs are better than defines

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> // get some infinity

// please no don't do this, example only
// change to value or notvalue

// #define SYMBOL notvalue
typedef long double HONKIN;

int main() {
    HONKIN value = 1e300;
    printf("%Lf\n", value);
    printf("%Le\n", value);
    value = INFINITY;
    printf("%Lf\n", value);
    printf("%Le\n", value);
    value = -INFINITY;
    printf("%Lf\n", value);
    printf("%Le\n", value);
    printf("HONKIN SIZE %lu\n", sizeof(value));
}

100000000000000005250476025520442024870446858110815915491585411551180245798890819578637
1.000000e+300
inf
inf
```

```
-inf
-inf
HONKIN SIZE 16
```

But typedefs are more meaningful

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> // get some infinity


// #define SYMBOL notvalue
typedef long long Count;
typedef double Measurement;


int main() {
    Count count = 1e9;
    Measurement m = 1.5;
    printf("Measurement %lf\n", m);
    printf("%lld\n", count);
    printf("Count SIZE %lu\n", sizeof(count));
    printf("Measurement SIZE %lu\n", sizeof(m));
}
```

```
Measurement 1.500000
1000000000
Count SIZE 8
Measurement SIZE 8
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> // get some infinity


// #define SYMBOL notvalue
typedef long long Count;
typedef double Measurement;

Count addCounts(Count count1, Count count2) {
    return count1 + count2;
}
Measurement addMeasurements(Measurement m1, Measurement m2) {
    return m1 + m2;
}

int main() {
    Count count1 = 2e9;
    Count count2 = 1e10;
    Measurement m1 = 1.5;
    Measurement m2 = 2.7;
    // this is intended
    printf("Measurement %lf\n", addMeasurements(m1,m2));
    printf("Count %lld\n", addCounts(count1,count2));

    // this shows the weakness of the lack of type checking on typedefs
    // we shouldn't mix counts and measurements
    printf("Measurement %lf\n", addMeasurements(m1,count1));
    printf("Count %lld\n", addCounts(count1,m2));

}
```

```
Measurement 4.200000
Count 12000000000
Measurement 2000000001.500000
Count 2000000002
```

1.6 Arrays

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

#define N 10

int main() {
    // initialization
    char notastring[N] = { 33, 34, 35, 37, 33,
                           32, 'a', 'b', 'c', 'd'};
    char astring[N] = "what!";
    // iterate over the notastring
    for ( int i = 0 ; i < N ; i++ ) {
        putchar( notastring[i ] );
    }
    putchar('\n');
    // iterate over the string terminating before we print \0
    for ( int i = 0 ; i < N ; i++ ) {
        if (astring[i] == '\0') {
            break;
        }
        putchar( astring[i] );
    }
    putchar('\n');
}
```

```
!"#%! abcd
what!
```

1.6.1 Array Initialization

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

#define N 10

int main() {
    // initialization
    char notastring[N];
    char anotherString[] = "!@#$%^&^%$#@#$%^&^%$#$%^&^%$#@$%^%$#@$%^%$#$%^&";
    // init the array programatically
    for ( int i = 0 ; i < N ; i++ ) {
        notastring[i] = 'a';
    }
    puts(notastring);
    puts("\n");
    puts(anotherString);
}
```

```
aaaaaaaaaa!@#$%^&^%$#@#$%^&^%$#$%^&^%$#@$%^%$#@$%^%$#$%^&


!@#$%^&^%$#@#$%^&^%$#$%^&^%$#@$%^%$#@$%^%$#$%^&
```

1.6.2 sizeof on arrays of characters

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

int main() {
    // initialization
    char notastring[] = {'a','b','c','d'};
    printf("notastring\t%ld\n",sizeof(notastring));
    printf("notastring[0]\t%ld\n",sizeof(notastring[0]));
    printf("char[1] \t%ld\n",sizeof(char[1]));
    printf("char[2] \t%ld\n",sizeof(char[2]));
    printf("char[4] \t%ld\n",sizeof(char[4]));
    printf("char[16]\t%ld\n",sizeof(char[16]));
    for ( int i = 0 ; i < sizeof( notastring ); i++ ) {
        putchar( notastring[i] );
    }
    putchar( '\n' );
}
```

```
notastring 4
notastring[0] 1
char[1]  1
char[2]  2
char[4]  4
char[16] 16
abcd
```

1.6.3 sizeof and integer arrays

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

int main() {
    // initialization
    int notastring[] = {'a', 1, 99, 128, 256, 512};
    printf("notastring\t%ld\n",sizeof(notastring));
    printf("notastring[0]\t%ld\n",sizeof(notastring[0]));
    printf("int[1] \t%ld\n",sizeof(int[1]));
    printf("int[2] \t%ld\n",sizeof(int[2]));
    printf("int[4] \t%ld\n",sizeof(int[4]));
    printf("int[16]\t%ld\n",sizeof(int[16]));
    for ( int i = 0 ; i < sizeof( notastring ); i++ ) {
        printf("%d\t", notastring[i]);
    }
    putchar( '\n' );
}
```

```
notastring 24
notastring[0] 4
int[1]  4
int[2]  8
int[4]  16
int[16] 64
97 1 99 128 256 512 152849920 -952870657 -1550235616 22002 94788503
32742 1 0 -648712648 32764 32768 1 -1550235910 22002 0 0 -1034247026
892415059
```

Oops something went wrong…
We're reading some memory that we shouldn't!
sizeof returns the total number of bytes, not the total entities

1.6.4 sizeof array versus sizeof element

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

int main() {
    // initialization
    int notastring[] = {'a', 1, 99, 128, 256, 512};
    printf("notastring\t%ld\n",sizeof(notastring));
    printf("notastring[0]\t%ld\n",sizeof(notastring[0]));
    printf("int[1] \t%ld\n",sizeof(int[1]));
    printf("int[2] \t%ld\n",sizeof(int[2]));
    printf("int[4] \t%ld\n",sizeof(int[4]));
    printf("int[16]\t%ld\n",sizeof(int[16]));
    for ( int i = 0 ; i < sizeof( notastring ) /  sizeof( notastring[0] ); i++ ) {
        printf("%d\t", notastring[i]);
    }
    putchar( '\n' );
}
```

```
notastring 24
notastring[0] 4
int[1]  4
int[2]  8
int[4]  16
int[16] 64
97 1 99 128 256 512
```

1.6.5 You must initalize arrays

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

#define N 5

void printIntArray( int n, int array[] ) {
    for (int i = 0 ; i < n ; i++) {
        printf("%d\t", array[i]);
    }
    printf("\n");
    return;
}

void example() {
    int values[N];
    printf("Before:\n");
    printIntArray( N, values );
    for (int i = 0; i < N; i++) {
        values[i] = 7 & (1+values[i]); // bitwise and
    }
    printf("After:\n");
    printIntArray( N, values);
}

int main() {
    example();
    printf("Coolbears\n");
    example();
    example();
}
```

```
Before:
-1200854624 32520 0 0 -1320310688
After:
1 1 1 1 1
Coolbears
Before:
1 1 0 0 -31551312
After:
2 2 1 1 1
Before:
2 2 1 1 1
After:
3 3 2 2 2
```

What happened?

1. Junk was left behind on the stack by the time the function executed. It was sucked into the example()'s values array.
2. The location of values is in the same spot on the stack if I call it the function again from the same location. The old values get reused.

1.6.6 Multidimensional arrays.

Arrays are blocks of memory allocated. Array locations are offsets of array entries are: sizeof(array[0])*i + &array

This means that a single dimensional array is a large block of memory allocated in order.

So an array of 32bit ints of size 10 will be 40 bytes long. The first entry is at offset 0 of the array, the next entry is at offset 4, the 10th entry is at 4*9 bytes away from the start of the array.

Multidimensional arrays are much the same.

2D arrays are sizeof(array[0]) * (row*cols + col) where row is the row number of a 2D array and col is the column number.

So a 2 rows x 3 cols will be allocated as:

int array[2][3];

The columns are stored together sequentially. The last column of row 0 is before the first column of row 1.

to access array[1][2] we would take the address of array and add the offset to it:

(array) + sizeof(array[0]) * (1 * (3) + 2)

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

#define N 5

void init2D(int rows, int cols, int values[rows][cols]) {
    int i = 0;
    for (int row = 0; row < rows; row++) {
        for (int col = 0; col < cols; col++) {
            values[row][col] = i++;
        }
    }
}
void print2D(int rows, int cols, int values[rows][cols]) {
   for (int row = 0; row < rows; row++) {
        for (int col = 0; col < cols; col++) {
            printf("%d\t", values[row][col]);
        }
        printf("\n");
    }
}
int main() {
    int values[N][N];
    init2D(  N, N, values );
    print2D( N, N, values );
}
```

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24
```

1.6.7 High dimensional Arrays

For 3dimensions or more we order the dimensions from back to from as an array of sizes SIZES[i] where 0 is the furtherest column and n-1 is the closest given:

int array[SIZES[n-1]][SIZES[n-2]]... [SIZES[0]]

given location per each dimenion as:

int location[n] = { . . . };
We calculate the location offset as
location[0]*SIZES[n-2]*SIZES[n-3]*. . . *SIZES[1]*SIZES[0] + . . . + location[n-3]*SIZES[1]*SIZES[0] + location[n-2]*SIZES[0] + location[n-1]
So watch out high dimensional arrays are a lot of multiplications.
int array[7][8][9];
array[6][5][4] is at
array + sizeof(int)*(6*8*9 + 5*9 + 4)

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

#define N 5

void init3D(int planes, int rows, int cols, int values[planes][rows][cols]) {
    int i = 0;
    for (int plane = 0; plane < planes; plane++) {
       for (int row = 0; row < rows; row++) {
          for (int col = 0; col < cols; col++) {
              values[plane][row][col] = i++;
          }
       }
    }
}
void print3D(int planes, int rows, int cols, int values[planes][rows][cols]) {
   for (int plane = 0; plane < planes; plane++) {
      printf("Plane: %d\n", plane);
      for (int row = 0; row < rows; row++) {
           for (int col = 0; col < cols; col++) {
               printf("%d\t", values[plane][row][col]);
           }
           printf("\n");
      }
   }
}
int main() {
    int values[N][N][N];
    init3D(  N, N, N, values );
    print3D( N, N, N, values );
}
```

```
Plane: 0
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
20 21 22 23 24
Plane: 1
25 26 27 28 29
30 31 32 33 34
35 36 37 38 39
40 41 42 43 44
45 46 47 48 49
Plane: 2
50 51 52 53 54
55 56 57 58 59
60 61 62 63 64
65 66 67 68 69
70 71 72 73 74
Plane: 3
75 76 77 78 79
80 81 82 83 84
85 86 87 88 89
90 91 92 93 94
95 96 97 98 99
Plane: 4
100 101 102 103 104
105 106 107 108 109
110 111 112 113 114
115 116 117 118 119
120 121 122 123 124
```

1.6.8 Variable Size Arrays

Only in C99+

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

void init2D(int rows, int cols, int values[][cols]) {
    int i = 0;
    for (int row = 0; row < rows; row++) {
        for (int col = 0; col < cols; col++) {
            values[row][col] = i++;
        }
    }
}
void print2D(int rows, int cols, int values[][cols]) {
   for (int row = 0; row < rows; row++) {
        for (int col = 0; col < cols; col++) {
            printf("%d\t", values[row][col]);
        }
        printf("\n");
    }
}
void example() {
    unsigned int n = 1 + rand() % 10;
    unsigned int m = 1 + rand() % 10;
    // unsigned int m = 10;
    // unsigned int n = 4;
    printf("%d X %d was chosen!\n", m, n);

    int values[m][n]; // SO the compiler can't predict this allocation ahead of time
    printf("sizeof(values) = %ld\n", sizeof(values));
    init2D(  m, n, values );
    print2D( m, n, values );
}
int main() {
    srand(time(NULL)); //initialze based on the clock
    example();
    example();
    example();
}
```

```
3 X 4 was chosen!
sizeof(values) = 48
0 1 2 3
4 5 6 7
8 9 10 11
5 X 10 was chosen!
```

```
sizeof(values) = 200
0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
9 X 8 was chosen!
sizeof(values) = 288
0 1 2 3 4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55
56 57 58 59 60 61 62 63
64 65 66 67 68 69 70 71
```

1.6.9 Exercises

- initialize an array of size 10 to 0 0 , 0 , 0 , 0, . . . , 0, 0

  int array[10]; for (i = 0; i < 10; i++) { array[i] = 0; }

- initialize an array of size 10 x 10 to 1 to 100

  1, 2, 3, . . . , 9, 10 . . . 91, . . . ,100

  int array[10][10]; int k = 1; for (int i = 0 ; i < 10; i++){ for (int j = 0 ; j < 10; j++){ array[i][j] = k++; } }

- initialize an array of size 10 x 10 to rows of 0,1,2,. . . ,9

  0,1,2,3,4,5 0,1,2,3,4,5 0,1,2,3,4,5

  int array[10][10]; for (int i = 0 ; i < 10; i++){ for (int j = 0 ; j < 10; j++){ array[i][j] = j; } }

  00000000 11111111 22222222

1.7 Functions

Functions replicate functions in mathematics. They allocate space on the stack and have local variables.

Very similar to python functions

Define a function:

$\text{return}_{\text{type}}$ functionName(ArgType1 arg1, ArgType2 arg2, ArgType3 arg3) { . . . }

Call a function:

functionName(arg1, arg2, arg3);

$\text{return}_{\text{type}}$ returnValue = functionName(arg1, arg2, arg3) ;

IN C89 all variable declarations are at the top of the function.

1.7.1 $\textbf{return}_{\textbf{types}}$

- void – nothing
- int
- char
- float
- double
- . . .
- pointer (array or string)

1.7.2 Example

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

void example() {
    printf("I have been made an example of\n");
}

int main() {
    example();
}
```

```
I have been made an example of
```

1.7.3 Pass by Value

The value of parameters are COPIED into registers and sometimes the stack. Thus the original variables that the parameters come from are safe.

Except pointers are not safe because given a pointer the called function can manipulate the data the pointer points to, but they cannot modify the original pointer.

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

int example(int x) {
    x++;
    return x;
}

int main() {
    int x = 10;
    printf("x: %d\n", x);
    int rx = example(x);
    printf("x: %d\n", x);
    printf("returned x vs x: %d vs %d\n", rx, x);
}
```

```
x: 10
x: 10
returned x vs x: 11 vs 10
```

1.7.4 Arrays again

- void initArray(int cols, int values[cols]) {
- void initArray(int cols, int values[]) {

You can specify array sizes in C99 but the size has to come earlier

- void init2D(int rows, int cols, int values[rows][cols]) {
- void init2D(int rows, int cols, int values[][cols]){
- void init3D(int planes, int rows, int cols, int values[planes][rows][cols]) {
- void init3D(int planes, int rows, int cols, int values[][rows][cols]) {

1.7.5 Don't trust sizeof inside of functions!

sizeof is only trustable if you declared the variable in your scope

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

void init2D(int rows, int cols, int values[][cols]) {
    int i = 0;
    printf("init2D: sizeof(values)=%lu\n", sizeof(values));
    printf("init2D: sizeof(values[0])=%lu\n", sizeof(values[0]));

    for (int row = 0; row < rows; row++) {
        for (int col = 0; col < cols; col++) {
            values[row][col] = i++;
        }
    }
}
void example() {
    unsigned int n = 1 + rand() % 10;
    unsigned int m = 1 + rand() % 10;
    printf("%d X %d was chosen!\n", m, n);
    int values[m][n]; // SO the compiler can't predict this allocation ahead of time
    printf("sizeof(values) = %ld\n", sizeof(values));
    printf("sizeof(&values) = %ld\n", sizeof(&values));
    printf("sizeof(values[0]) = %ld\n", sizeof(values[0]));
    init2D(  m, n, values );
}
int main() {
    srand(time(NULL)); //initialze based on the clock
    example();
    example();
    example();
}
```

```
1 X 4 was chosen!
sizeof(values) = 16
sizeof(&values) = 8
sizeof(values[0]) = 16
init2D: sizeof(values)=8
```

```
init2D: sizeof(values[0])=16
4 X 5 was chosen!
sizeof(values) = 80
sizeof(&values) = 8
sizeof(values[0]) = 20
init2D: sizeof(values)=8
init2D: sizeof(values[0])=20
7 X 3 was chosen!
sizeof(values) = 84
sizeof(&values) = 8
sizeof(values[0]) = 12
init2D: sizeof(values)=8
init2D: sizeof(values[0])=12
```

1.7.6 Returns

Don't return arrays in general.

To return a value and exit the function immediately run:

return expr

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

int squareInt(int x) {
    return x*x;
}

float squareFloat(float x) {
    return x*x;
}
int intDiv(int x, int y) {
    return x/y;
}
float floatDiv(float x, float y) {
    return x/y;
}
char returnChar( int i ) {
    return i;
}
```

```
int main() {
    printf("squareInt\t %d\n", squareInt(25));
    printf("squareInt\t %d\n", squareInt(1.47));
    printf("squareFloat\t %f\n", squareFloat(1.47));
    printf("squareFloat\t %f\n", squareFloat(25));
    printf("intDiv\t %d\n", intDiv(64,31));
    printf("intDiv\t %d\n", intDiv(64.2,31));
    printf("floatDiv\t %f\n", floatDiv(64,31));
    printf("floatDiv\t %f\n", floatDiv(64.2,31));
    printf("returnChar\t %hhu\n", returnChar( 578 ) );
    printf("returnChar\t %hhu\n", returnChar( 'a' ) );
    printf("returnChar\t %hhu\n", returnChar( 66.1 ) );
    printf("returnChar\t %c\n", returnChar( 578 ) );
    printf("returnChar\t %c\n", returnChar( 'a' ) );
    printf("returnChar\t %c\n", returnChar( 66.1 ) );

}
```

```
squareInt  625
squareInt  1
squareFloat  2.160900
squareFloat  625.000000
intDiv  2
intDiv  2
floatDiv  2.064516
floatDiv  2.070968
returnChar  66
returnChar  97
returnChar  66
returnChar  B
returnChar  a
returnChar  B
```

1.7.7 Recursion

1. Recursion

   (a) Recursion

      i. Recursion

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

```
#include <stdlib.h>

int divisibleBy(int x, int y) {
    printf("%d %d\n", x,y);
    if (x == 0) { return 0; }
    if (y <= 0) { return 0; }
    if (x % y == 0) { return y; }
    return divisibleBy(x, y - 1);
}

int main() {
    printf("%d\n",divisibleBy(77,76));
}
```

```
77 76
77 75
77 74
77 73
77 72
77 71
77 70
77 69
77 68
77 67
77 66
77 65
77 64
77 63
77 62
77 61
77 60
77 59
77 58
77 57
77 56
77 55
77 54
77 53
77 52
77 51
```

```
77  50
77  49
77  48
77  47
77  46
77  45
77  44
77  43
77  42
77  41
77  40
77  39
77  38
77  37
77  36
77  35
77  34
77  33
77  32
77  31
77  30
77  29
77  28
77  27
77  26
77  25
77  24
77  23
77  22
77  21
77  20
77  19
77  18
77  17
77  16
77  15
77  14
77  13
77  12
77  11
```

```
11
```

1.7.8 Prototypes

1.7.9 Exercise

- make a recursive countdown function, printing each number until 0 is met.
- make a recursive fibonacci