02 - Lecture - C language basics

Data types in C

char <= short <= int <= long <= long long

- C standard does not specify byte sizes of these types.
- on most systems:

    char is 1 byte, short is 2, int is 4, long long is 8.

- long is the problematic one:

    Most 32-bit systems and 64-bit Windows: 4 bytes
    64-bit UNIX (such as our Linux system): 8 bytes
    Java: 8 bytes

- if you need to ensure byte sizes, use int<N>_t types:
    - int8_t, int16_t, int32_t, int64_t
    - #include <stdint.h>
    - defined in C99 standard
- two’s complement system
- also serve as boolean
- examples of integer variable decralations:

```
int x;
int x, y;
int x = 0, y;

char c = 'x';
char c = '\n';
char c = '\13';

char c = '0';
char c = '\0';
char c = 0;

long x = 0L;
```

unsigned version of all of the above

```
unsigned long x = 0, y = 0xff00ff00ff00ff00UL

uint32_t x = 0xffffffff
```

float is 4 bytes and double is 8 bytes

```
123.4f
123.4
```

arrays and pointers

no strings!

Expressions

literals and variables

function calls

assignment:

```
    lvalue = rvalue
```

pre/post-inc/decrement

```
    x = i++;
    x = ++i;
```

operations

```
    arithmetic:   +, -, *, /, %
    comparison:   <, >, ==, !=, <=, >=
    logical:      &&, ||, !
    bitwise:     ~, &, |, ^, <<, >>
```

- assignement versions of arithmetic and bitwise ops
- short-circuit eval in logical ops

comman expression

conditional expression (ternary operator)

```
    z = (a > b) ? a : b;

    z = max(a, b);
```

any integral expression is also a boolean expression

Statements

if-else:
- which if does else bind to?

switch:
- another form of else-ifs.
- don’t forget "break;"!

loops:
- for, while, do-while
- memorize idioms for looping from 0 to n-1 (i.e., n times)
- break, continue

goto
  - not as evil as you might have heard

Variable scoping

```
int x;
x = 0;

{
    int x;
    x = 1;
    printf("%d", x);
}

printf("%d", x);
```

Storage class
----

1) automatic variables

  - also called stack variables, since they are usually stored in process stack (we'll see what this means later)

  - scope: local to a block

  - lifetime: created on block entrance, destroyed on exit

  - example:

```
int foo(int auto_1)
{
    int auto_2;

    {
        int auto_3;

        ...
    }

    ...
}
```

2) static variables

  - "static" has so many meanings in C/C++/Java, so brace yourself!

  - stored in global data section of process memory

  - scope depends on where it is declared: global, file, or block

  - lifetime: created and initialized on program start-up, and persists until the program ends

  - example:

```
int global_static = 0;  // visible to other files

static int file_static = 0;  // only visible within this file

int foo(int auto_1)
{
    static int block_static = 0; // only visible in this block

    ...
}
```

Definition and declaration of global variables
---

1) *defining* a global variable:

```
int x = 0;

extern int x = 0;
```

2) *declaring* a global variable that is defined in another file:

```
extern int x;
```

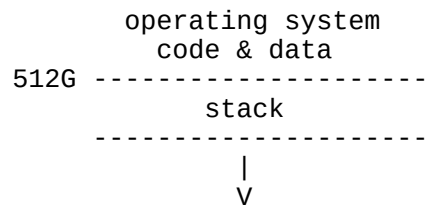
3) defining a global variable *tentatively*

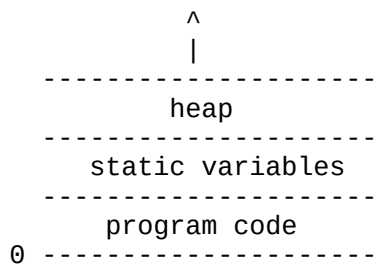
```
int x;
```

- same as "int x = 0;" if no other definition of x appears in the same file
- same as "extern int x;" if something like "int x = 5;" appears in the same file
- the moral of the story is: don’t do it!

Process address space

Every single process (i.e., a running program) gets 512GB of memory space:





Obviously, computers don’t have that much RAM. It’s virtual memory!