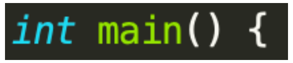
COMPSCI2030 Systems Programming

Components of a C Program

Yehia Elkhatib

University of Glasgow

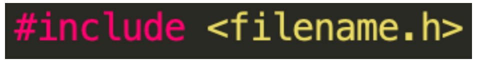
main()



```
int main() {
```

- The only one required for *any* C program
- It denotes the entry point to the program
  - there can only be one and exactly one
- It returns an int to signify the exit code
  - `0` = normal execution and termination, i.e. at last statement in main
  - Non Zero Exit Code = abnormal termination
  - if no `return` statement, an implicit `return 0` is executed
- Can be used to pass command line arguments – next lecture

#include

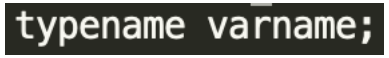


```
#include <filename.h>
```

- Instructs the compiler to add contents of a file to your program
  - they are included as is, i.e. you do not need to modify their logic
- Include files are usually called *header* files
  - pre-existing libraries, e.g. `#include <stdio.h>`
  - user-defined , e.g. `#include “myheader.h”`
- Commonly used library header files
  - stdio.h
  - stdlib.h
  - string.h
  - limits.h

Variables

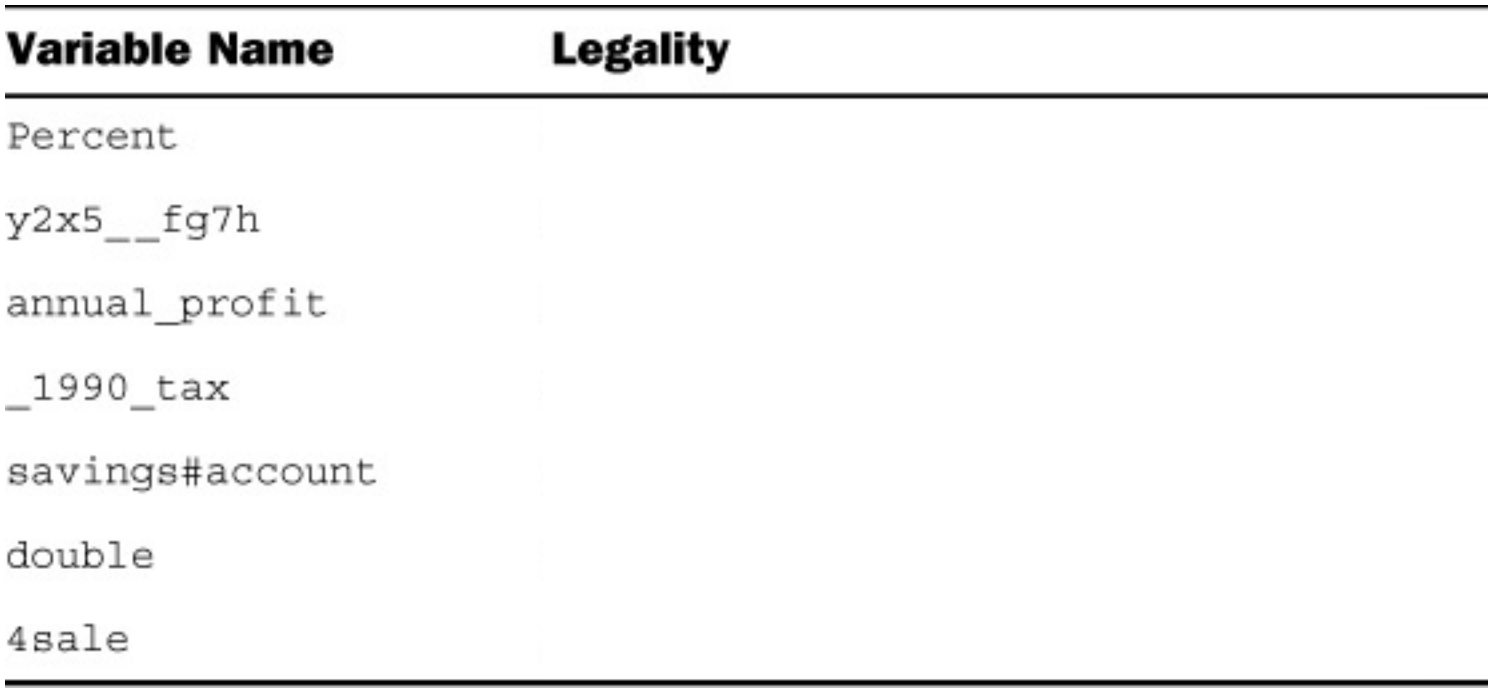
- Variable – name assigned to a location in memory to store data
- Have to be defined before using, informing the compiler of:
  - variable name
  - data type



```
typename varname;
```

- Names
  - can contain letters, digits, and underscores
  - must start with a letter (underscore also accepted, but not recommended)
  - <u>case-sensitive</u>
  - must not be a reserved keyword; e.g. int, return, sizeof
- C is statically typed => every variable/expression has to have a **data type** that is known without running the program

Variables



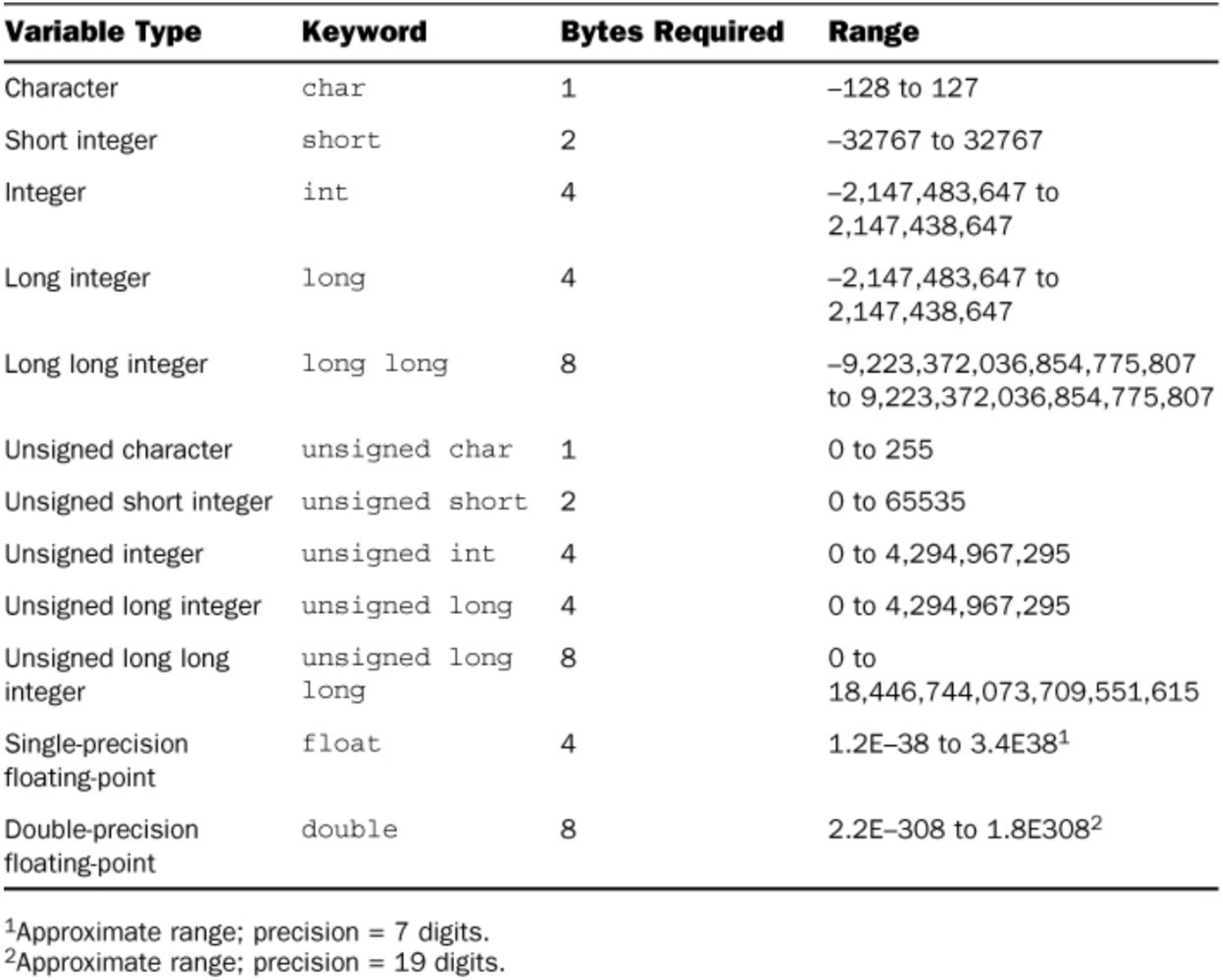
Variable Name	Legality
Percent	
y2x5__fg7h	
annual_profit	
_1990_tax	
savings#account	
double	
4sale	

Data Types

- What is the meaning of the bit-pattern `1000 0001` ?
    - maybe: 129 if it represents an unsigned 8-bit integer value
    - maybe: -127 if it represents a signed 8-bit integer value (2’s compl.)
    - maybe: the colour blue? or an ASCII character? … etc.
- The programmer gives meaning to a collection of bits
- The computer needs a way to identify different types of meanings
- By declaring a variable with a certain data type we decide what the bit-pattern in memory means

Data Types

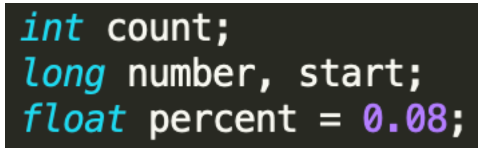
- By choosing a particular data type, we control how much memory we use



Variable Type	Keyword	Bytes Required	Range
Character	`char`	1	–128 to 127
Short integer	`short`	2	–32767 to 32767
Integer	`int`	4	–2,147,483,647 to 2,147,438,647
Long integer	`long`	4	–2,147,483,647 to 2,147,438,647
Long long integer	`long long`	8	–9,223,372,036,854,775,807 to 9,223,372,036,854,775,807
Unsigned character	`unsigned char`	1	0 to 255
Unsigned short integer	`unsigned short`	2	0 to 65535
Unsigned integer	`unsigned int`	4	0 to 4,294,967,295
Unsigned long integer	`unsigned long`	4	0 to 4,294,967,295
Unsigned long long integer	`unsigned long long`	8	0 to 18,446,744,073,709,551,615
Single-precision floating-point	`float`	4	1.2E–38 to 3.4E38[1]
Double-precision floating-point	`double`	8	2.2E–308 to 1.8E308[2]

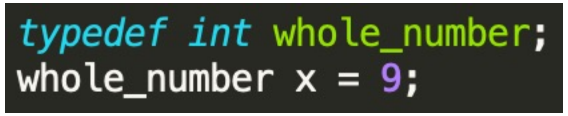
[1] Approximate range; precision = 7 digits.
[2] Approximate range; precision = 19 digits.

Variable Declarations



```
int count;
long number, start;
float percent = 0.08;
```

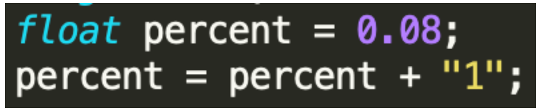
- Before using a variable, it must be declared
- A variable declaration tells the compiler the name and type
  - The declaration may also initialize the variable to a specific value
- Using an undeclared variable throws a compiler error
- Variables are stored at locations in memory that do not change over their lifetimes (explained in the next lecture)
- `typedef` creates a new name for an existing data type
  - essentially creating a synonym
  - typically used with structs



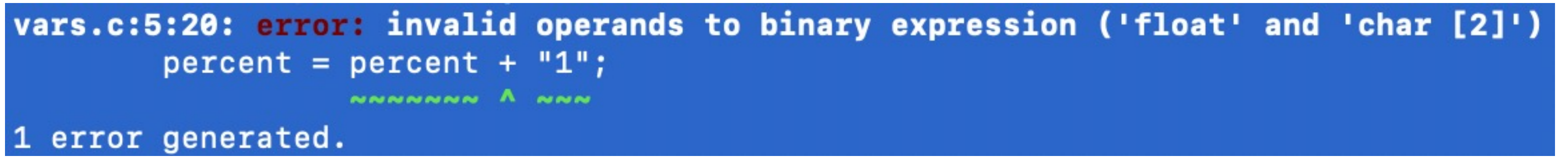
```
typedef int whole_number;
whole_number x = 9;
```

Statically Typed Variables

- To assist us writing *meaningful programs* the compiler enforces that computations *preserve the meaningful representation of our data*
  - e.g. for x+1 the compiler ensures that a *meaningful* addition of the value one and x is performed
- By enforcing operations to respect data types, the compiler prevents meaningless computations



```
float percent = 0.08;
percent = percent + "1";
```



```
vars.c:5:20: error: invalid operands to binary expression ('float' and 'char [2]')
        percent = percent + "1";
                  ~~~~~~~ ^ ~~~
1 error generated.
```

Lab Sheet

Task 2.A

```
for (i = 0; i < 3; i++) {
    unsigned int cp_count;
    unsigned int len = n;
    if (user(group_info[i]){
        return -EFAULT;
    grouplist += NGS_PER_BLOCK;
    count -= cp_count;
```

```
group_info = kmalloc(usr)
if (!group_info)
    return NULL;
group_info->ngroups = gidsetsize;
group_info->nblocks = nblocks;
atomic_set(&group_info->usage, 1)
```

```
if (gidsetsize <= NGROUPS_SMALL)
    group_info->blocks[0] = gr;
else {
    for (i = 0; i < nb; i++) {
        gid_t *b;
        b = (void *)__get(GFP);
        if (!b)
            goto partial_alloc;
        group_info->blo[i] = b;
    }
```

```
struct group_inf
int nblocks;
```

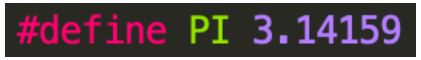
```
void groups_free(struct=) {
    if (group_info->blocks[0]) {
        int i;
        for (i = 0; i < 1; i++)
            free_page();

    e(group_info);
```

Boolean Variables

- In C every integer can be interpreted as a boolean, where `0` represents false and any other value true
- In C99 standard, `_Bool` was added as a data type
- Is an `unsigned int`
- Values: `0`, `1`
- Takes 1 bit of memory
- Alternative: bool from stdbool.h
  - values: `true`, `false`

Symbolic Constants – two ways

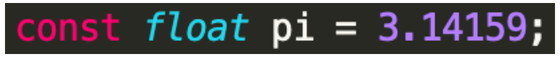
- The `#define` directive



```
#define PI 3.14159
```

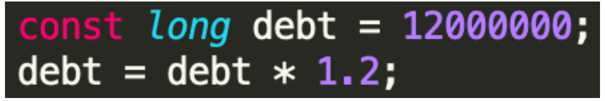
  - *“hey compiler, find and replace each of these names with this value”*
  - notice: no semicolon
  - by convention, names are uppercase so they are easy to distinguish
  - by convention, group all `#define` statements before the `main()` function
    - can be placed anywhere, but a constant is only valid for code that follows its `#define`

- The `const` keyword

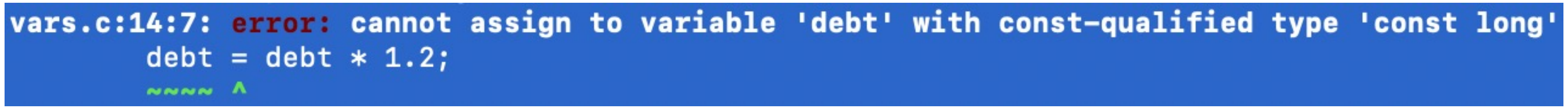


```
const float pi = 3.14159;
```

  - a modifier that can be applied to any variable declaration
  - a value initialized at declaration is prohibited from being changed later



```
const long debt = 12000000;
debt = debt * 1.2;
```



```
vars.c:14:7: error: cannot assign to variable 'debt' with const-qualified type 'const long'
        debt = debt * 1.2;
        ~~~~ ^
```

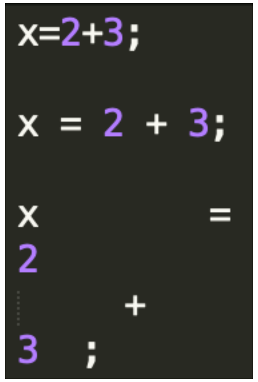
Lab Sheet

Task 2.B

for (i = 0; i < 3; i++) {
unsigned int cp_count;
unsigned int len = n;
if (user(group_info[i]){
return -EFAULT;
grouplist += NGS_PER_BLOCK;
count -= cp_count;
group_info = kmalloc(usr)
if (!group_info)
return NULL;
group_info->ngroups = gidsetsize;
group_info->nblocks = nblocks;
atomic_set(&group_info->usage, 1)
if (gidsetsize <= NGROUPS_SMALL)
group_info->blocks[0] = gr;
else {
for (i = 0; i < nb; i++) {
gid_t *b;
b = (void *)__get(GFP);
if (!b)
goto partial_alloc;
group_info->bio[i] = b;
}
struct group_inf
int nblocks;
void groups_free(struct=) {
if (group_info->blocks[0]) {
int i;
for (i = 0; i < 1; i++)
free_page();
(group_info);

Statements

- An instruction that directs the computer to carry out some task
- End with a semicolon
  - except for preprocessor directives such as `#define` and `#include`
- Are not white-space sensitive

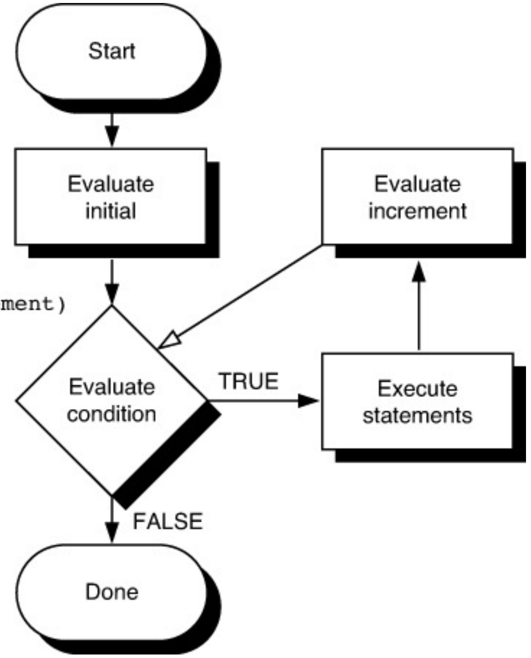


```
x=2+3;

x = 2 + 3;

x        =
2
     +
3  ;
```

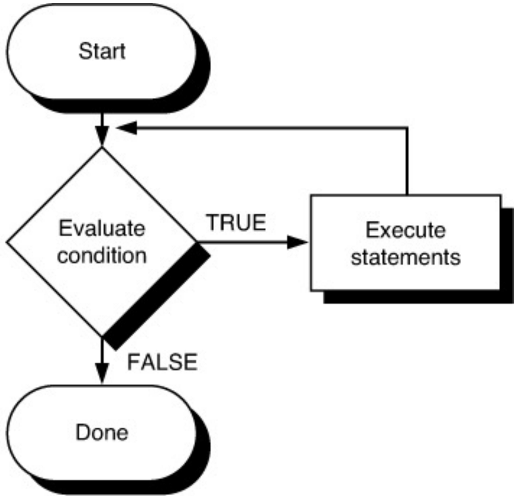
Loops

- `for`, `while`, `do-while`
- Syntax similar to Java

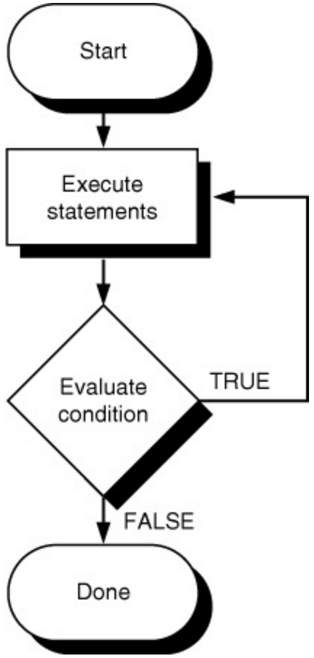
```
for (initial; condition; increment)
    statements
```



```
while (condition)
        statements;
```

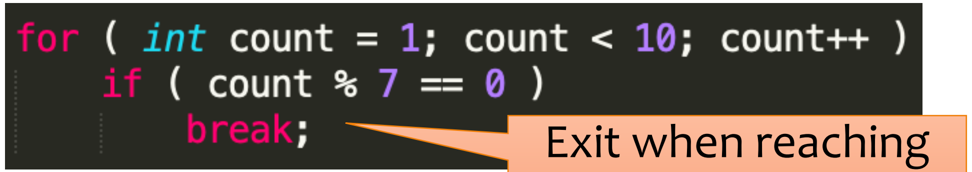


```
do statements;
while (condition);
```



Ending loops early

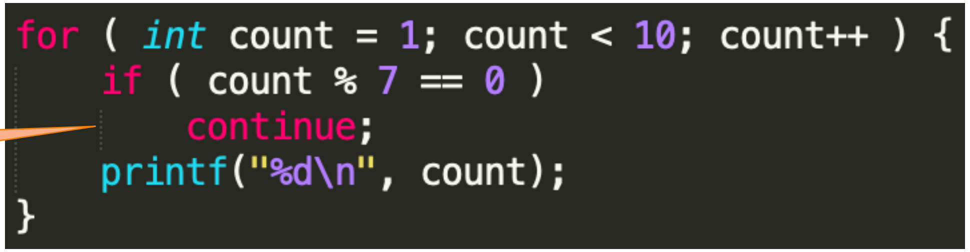
- You might want to exert more control over loop execution
- break
  - could be used with for, while, do-while (and switch)
  - execution immediately exits the loop



```
for ( int count = 1; count < 10; count++ )
    if ( count % 7 == 0 )
        break;
```

Exit when reaching a multiple of 7

- continue
  - the next iteration of the enclosing loop begins
  - statements between continue and the end of the loop are <u>not</u> executed

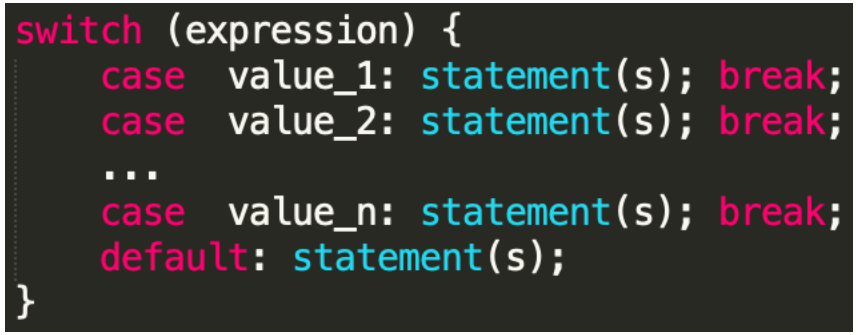


```
for ( int count = 1; count < 10; count++ ) {
    if ( count % 7 == 0 )
        continue;
    printf("%d\n", count);
}
```

Print all integers but multiples of 7

switch

- Lets you execute different statements based on an expression
- Useful when the expression can have more than 2 values
  - `if` is limited to evaluating an expression as true or false



```
switch (expression) {
    case  value_1: statement(s); break;
    case  value_2: statement(s); break;
    ...
    case  value_n: statement(s); break;
    default: statement(s);
}
```

- If a match is found between expression and one of the values: execution is transferred to the statement that follows the case label
- Otherwise, execution is transferred to the statement following the <u>optional</u> default label

Lab Sheet

Task 2.C

for (i = 0; i < 3; i++) {
unsigned int cp_count;
unsigned int len = n;
if (user(group_info[i]){
return -EFAULT;
grouplist += NGS_PER_BLOCK;
count -= cp_count;
group_info = kmalloc(usr)
if (!group_info)
return NULL;
group_info->ngroups = gidsetsize;
group_info->nblocks = nblocks;
atomic_set(&group_info->usage, 1)
if (gidsetsize <= NGROUPS_SMALL)
group_info->blocks[0] = gr;
else {
for (i = 0; i < nb; i++) {
gid_t *b;
b = (void *)__get(GFP);
if (!b)
goto partial_alloc;
group_info->blo[i] = b;
}
struct group_in
int nblocks;
void groups_free(struct=) {
if (group_info->blocks[0]) {
int i;
for (i = 0; i < 1; i++)
free_page();
(group_info);

Components of a program – to be continued

- Functions
- … in the next lecture

Question time

1. Why not always use the larger variables, such as `long int` and `double` instead of `int` and `float` to hold bigger numbers?
2. What happens if you put a number into a type that is not big enough to hold it?
3. In what variable type would you best store the following values?
   - The number of Facebook friends a person has
   - The radius of a circle
   - Your annual salary
   - A person’s first initial
   - The distance to a star in miles