# C Programming 1DT301

Lecture 1

[OPNOVA]
ENGINEERED INNOVATION

# Content

- C (Minimal basics)
- Useful constructs when programming in small embedded systems
- No Analysis & Design!

# Litterature

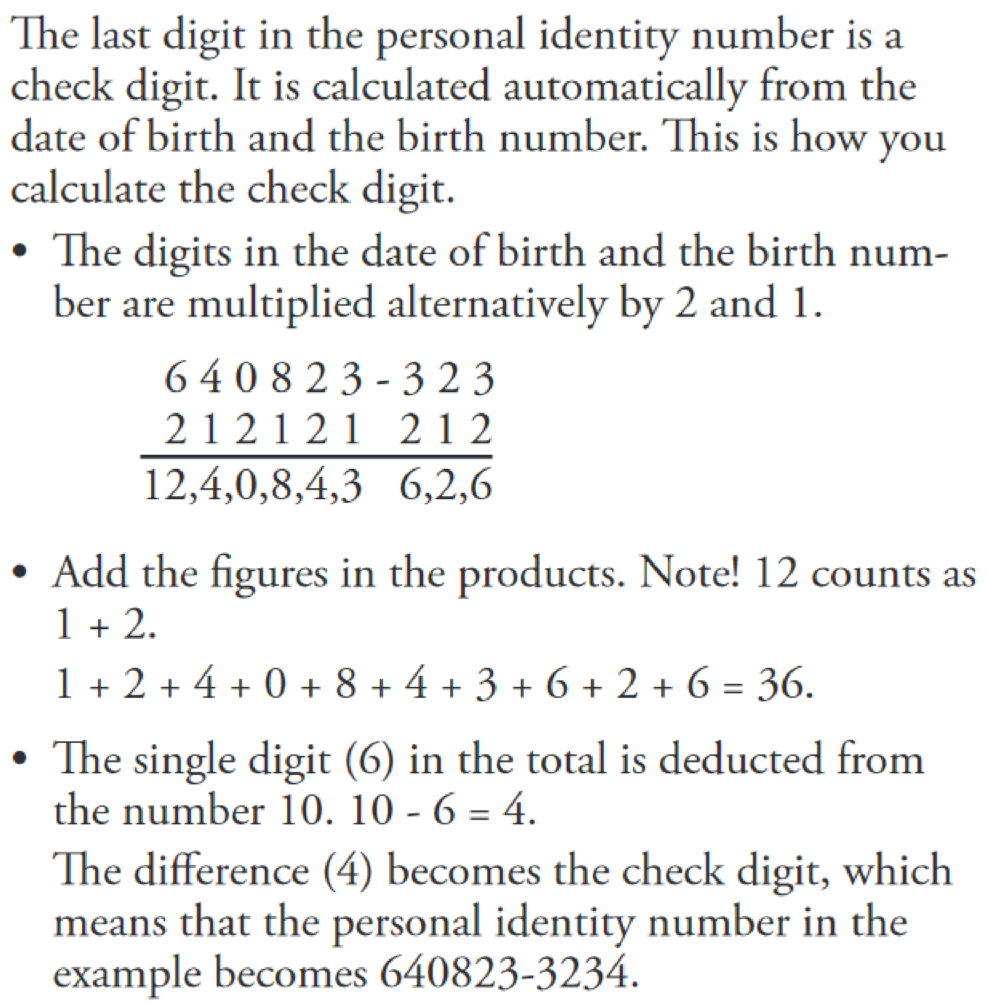
- The C Book, http://publications.gbdirect.co.uk/c_book/thecbook.pdf
- Learning GNU C, http://nongnu.askapache.com/c-prog-book/learning_gnu_c.pdf
- The GNU C Reference Manual, http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf

# Programming-In-General

- Think first → Program later
  - In general don’t use Trial-by-error (or Learning-by-doing)
- Mental image of things & Algorithmic thinking
  - Building a house where others would live...
- Abstraction → Concrete

*(Have a look at the concept of Computational Thinking)*

# Example of an algorithm



The last digit in the personal identity number is a check digit. It is calculated automatically from the date of birth and the birth number. This is how you calculate the check digit.

- The digits in the date of birth and the birth number are multiplied alternatively by 2 and 1.

```
  6 4 0 8 2 3 - 3 2 3
  2 1 2 1 2 1   2 1 2
 ---------------------
 12,4,0,8,4,3   6,2,6
```

- Add the figures in the products. Note! 12 counts as 1 + 2.

  1 + 2 + 4 + 0 + 8 + 4 + 3 + 6 + 2 + 6 = 36.

- The single digit (6) in the total is deducted from the number 10. 10 - 6 = 4.

  The difference (4) becomes the check digit, which means that the personal identity number in the example becomes 640823-3234.

There is no programming language here!

http://www.skatteverket.se/privat/folkbokforing/omfolkbokforing/personnumretsuppbyggnad.4.18e1b10334ebe8bc80001502.html

# C

- Small & terse language (few keywords and constructs)
- Low-level (mostly)
- “Subset” of C++
- UNIX
- Dennis Ritchie[1]
- ALGOL60 → BCPL→ B → C[2]

[1] cm.bell-labs.com/cm/cs/who/dmr (search for `The Development of the C Language')

[2] http://www.levenez.com/lang/

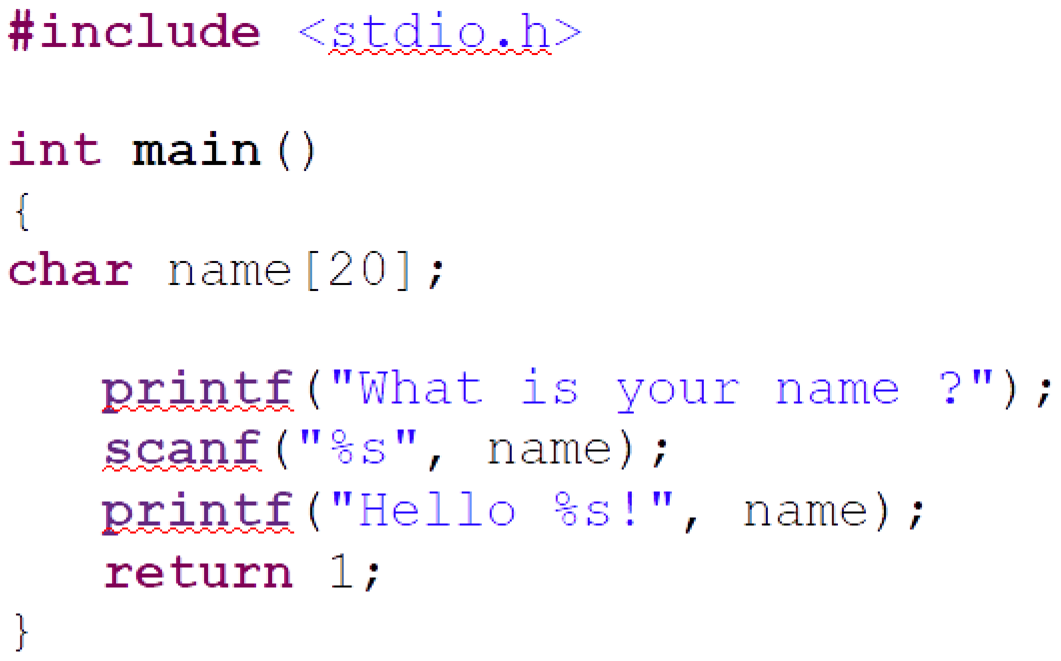
# Typical Uses

- Real-time systems
- Embedded systems
- Server applications
- Desktop applications (console)

# Keywords

- auto
- break
- case
- char
- const
- continue
- default
- do
- double
- else
- enum
- extern
- float
- for
- goto
- if
- inline
- int
- long
- register
- restrict
- return
- short
- signed
- sizeof
- static
- struct
- switch
- typedef
- union
- unsigned
- void
- volatile
- while
- _Bool
- _Complex
- _Imaginary

# A C Program

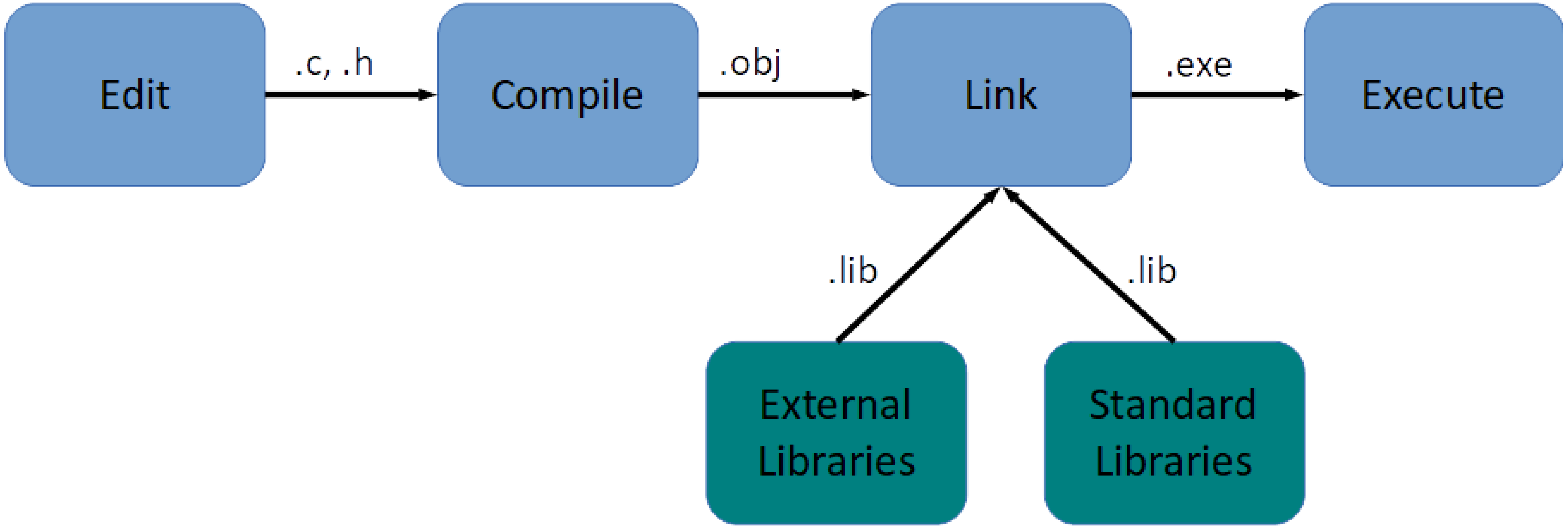


```
#include <stdio.h>

int main()
{
char name[20];

   printf("What is your name ?");
   scanf("%s", name);
   printf("Hello %s!", name);
   return 1;
}
```

# Workflow



There is more to it than this!

# Declaration vs. Definition

- Declaration
  - A place where a symbol (name+type) is stated.
  - Compiler can do its job with declarations only.
- Definition
  - The place where the symbol is created or assigned storage.
  - Linker needs definitions.
  - Missing definitions will produce (weird) linker errors!
- An identifier can be both declared and defined at the same time.

# Headers

- Contains declarations
  - So there need to be definitions somewhere!
- AKA *Include files* (and similar Assembly language `.include` directive).
- Standard libraries (need .lib-files!)
- Modularisation[1]
- Abstract Data Types[1]

[1] Not in this course.

# Headers

- File (.h)
- Referred to by using #include
- #include <stdio.h> ↔ #include ”stdio.h”
- Custom made header files.
  - Typically #include ”filename.h”
- Problem with circular includes.

# Some standard header files

| Include | Content |
|---|---|
| assert.h | Diagnostics. |
| math.h | Mathematical functions and macros |
| stdio.h | Input and output functions and macros. |
| stdlib.h | Number conversions, storage allocations etc… |
| string.h | String handling. |
| time.h | Manipulating time and date. |

# Call-By-Value

- When calling a function, You receive a copy of the caller's value(s).
- Simulates Call-By-Reference by using pointers.

# Preprocessor

- Performs
  - Inclusion of files (#include)
  - Macro substitution. This means textual replacement! Watch out!
  - Conditional compilation
- Starts with the # character

(More on this later)

# Variables & Types

# Variable

- *A reserved place in memory*
- Identifier (name)
- Type (size !)

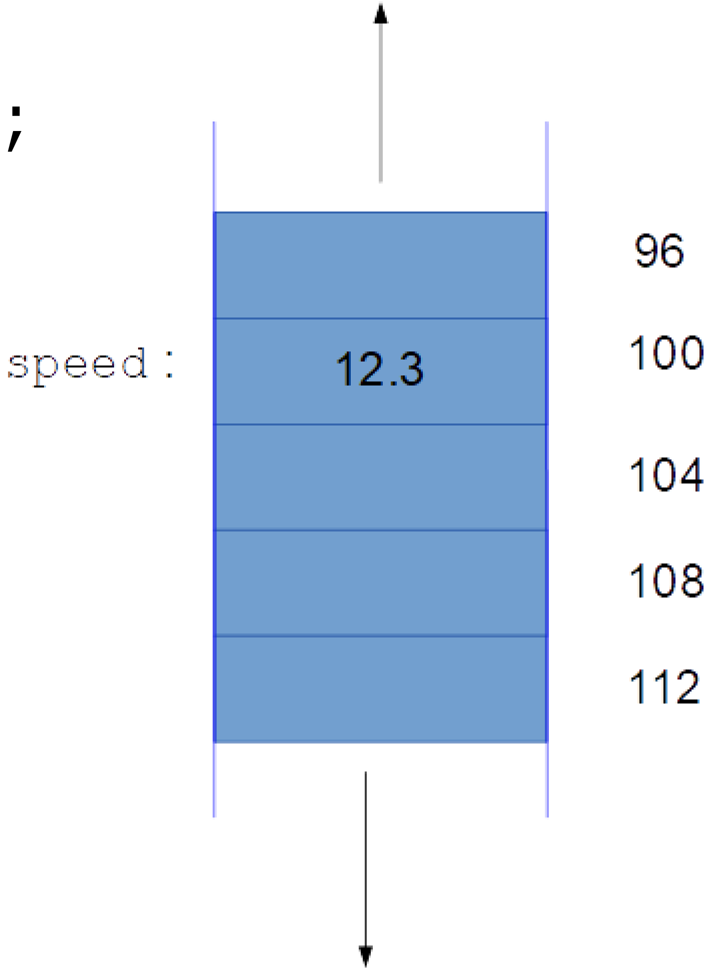
Type

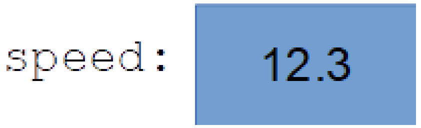
Identifier

```
float speed;
speed = 12.3;
```

# Variables

```
float speed;
speed = 12.3;
```





***Think in terms of memory locations and spaces!***

# Scope

- Life of a variable (identifiers).
- Storage classes
  - auto, register, extern, static

# Constants

- Not a variable :)
- Prevent from overwriting (hopefully...)
- const int MYCONST = 23;
- #define MYCONST 23

# Enumeration Constants

- A set of integers represented by identifiers.
  - Instead of a separate const declarations.
- User-defined type.

```
enum card {CLUBS, HEARTS, SPADES, DIAMONDS};

enum card {CLUBS = 0, HEARTS, SPADES, DIAMONDS};

enum card {CLUBS, HEARTS=2, SPADES = 7, DIAMONDS};
```

# Types

- Implementation dependent
  - `int` on one vendor is implemented as 16-bits, another vendor uses 8-bits.
- Misuse of → Many times the result is a *Undefined behavior.*
- `void` - means 'an object having a non-existing value'.

# Some Types (there are more!)

| Type | Description | Range |
|---|---|---|
| char | Usually a byte character. | -128 to 127 |
| unsigned char | A byte | 0 to 255 |
| short, int | At least a 16-bit integer | –32,768 to 32,767 |
| unsigned short, unsigned int | If 16-bit → a word. | 0 to 65,535 |
| float | Single precision floating point (4 bytes). | 3.4E +/- 38 (7 digits) |
| double | Double precision floating point (8 bytes). | 1.7E +/- 308 (15 digits) |

Note! Implementation dependent.

Note! Limits are defined in limits.h.

# ASCII

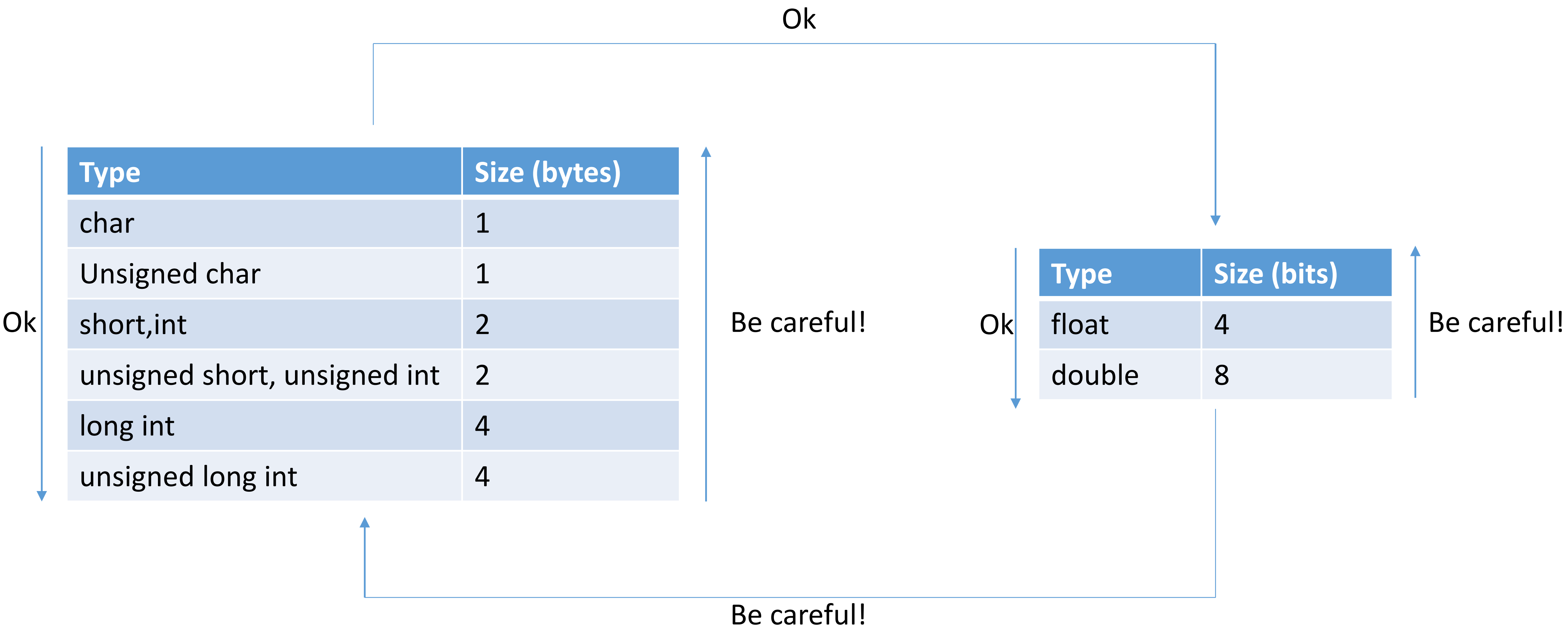
- char code value (an internal representation)
- An integer (byte)
- 0-255

| Value | Character | Control |
|---|---|---|
| 10 | - | LF (Line Feed) |
| 12 | - | CR (Carriage Return) |
| 65 | A | |
| 97 | a | |

# Casting

- Converting from one type to another.
- Implicit/automatic
  - Different operands, assignment, function arguments, return value from a function.
- Explicit
  - (type)expression

# Casting



Ok

| Type | Size (bytes) |
|---|---|
| char | 1 |
| Unsigned char | 1 |
| short,int | 2 |
| unsigned short, unsigned int | 2 |
| long int | 4 |
| unsigned long int | 4 |

Ok

Be careful!

| Type | Size (bits) |
|---|---|
| float | 4 |
| double | 8 |

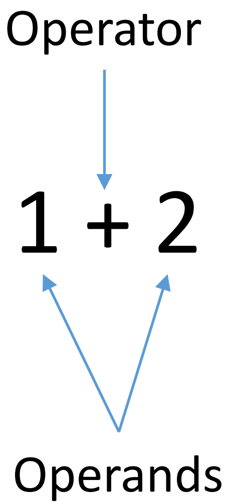
Ok

Be careful!

Be careful!

# Operators

# Operand vs. Operator



Operator

1 + 2

Operands

# Expressions

- They return values (statemens don’t)

# Assignment

| Operator | Meaning | Comment |
|---|---|---|
| = | Assign | a=2 |
| += | Add and assign | a+=2 → a=a+2 |
| -= | Subtract and assign | a-=2 → a=a-2 |
| *= | Multiply and assign | a*=2 → a=a*2 |
| /= | Divide and assign | a/=2 → a=a/2 |

There are more assignment operators!

# Arithmetic

| Operator | Meaning | Comment |
|---|---|---|
| + | Add | 1+2 |
| - | Subtract | 1-2 |
| * | Multiply | 1*2 |
| / | Divide | 9.0/2 → 4.5<br>9/2 → 4<br><br>Note! If operands are integers → Integer division! |
| % | Modulo | 4 % 2 |

# Precedence

| Order | Operator | Comment |
|---|---|---|
| 1 | () | Force order of evaluation. |
| 2 | * / % | Left to right. |
| 3 | + - | Left to right. |

$$y = \frac{2x - 1}{5}$$ → y=(2x-1)/5

# Increment/Decrement (Unary)

- ++ → increment
- -- → decrement
- Prefix
  - Increment/decrement before its value is used
- Postfix
  - Increment/decrement after its value is used

# Boolean

- _Bool
- Is an integer.
- 0 → False
- Everything else → true
- Standard dependent
  - Exists as a type in newer versions (bool).

# Relational

| Operator | Meaning | Comment |
|---|---|---|
| < | Less than | 1<2 → 1 (true) |
| <= | Less than or equal | 1<=2 → 1 (true)<br>2<=2 → 1 (true) |
| > | Greater than | 1 > 2 → 0 (false) |
| >= | Greater than or equal | 1 >= 2 → 1 (true)<br>2 >= 2 → 1 (true) |
| == | Is equal to | 1 == 1 → 1 (true)<br>1 == 0 → 0 (false) |
| != | Not equal to | 1 != 1 → 0 (false)<br>1 != 0 → 1 (true) |

*Be Warned*! Testing for equality (or not) when operands are of floating point type is not recommended !!

# Floating points and Equality

- Problem since floats (sometimes) aren't represented exactly.
- Use epsilons when testing for equality !
  - FLT_EPSILON
  - DBL_EPLSILON

Include float.h

Example

# Logical (not Bitwise!)

| Operator | Meaning | Comment |
|---|---|---|
| && | And | 1 && 0 → 0 |
| \|\| | Or | 1 \|\| 0 → 1<br>1 \|\| 2 → 1 |
| ! | Not | !1 → 0<br>!0 → 1 |

Remember

0 → False
Anything else → True

# Logical Operators and Short Circuiting

- Logical operators are read *from left to right*.
- If an operator to the left is false, then the operators to the right won't be evaluated ( e.g. if they are functions!).

```
int a = 1, b = 0, c = 0;

If( c && (a || b))
{
  // Some code
}
```

|| will not be evaluated!

# Example

- Create a program that allows a user to add two integers. The sum is printed in the screen.