INSTITUT POLYTECHNIQUE DE PARIS

Variables and types

Bachelor of Science - École polytechnique

gael.thomas@inria.fr

Key concepts

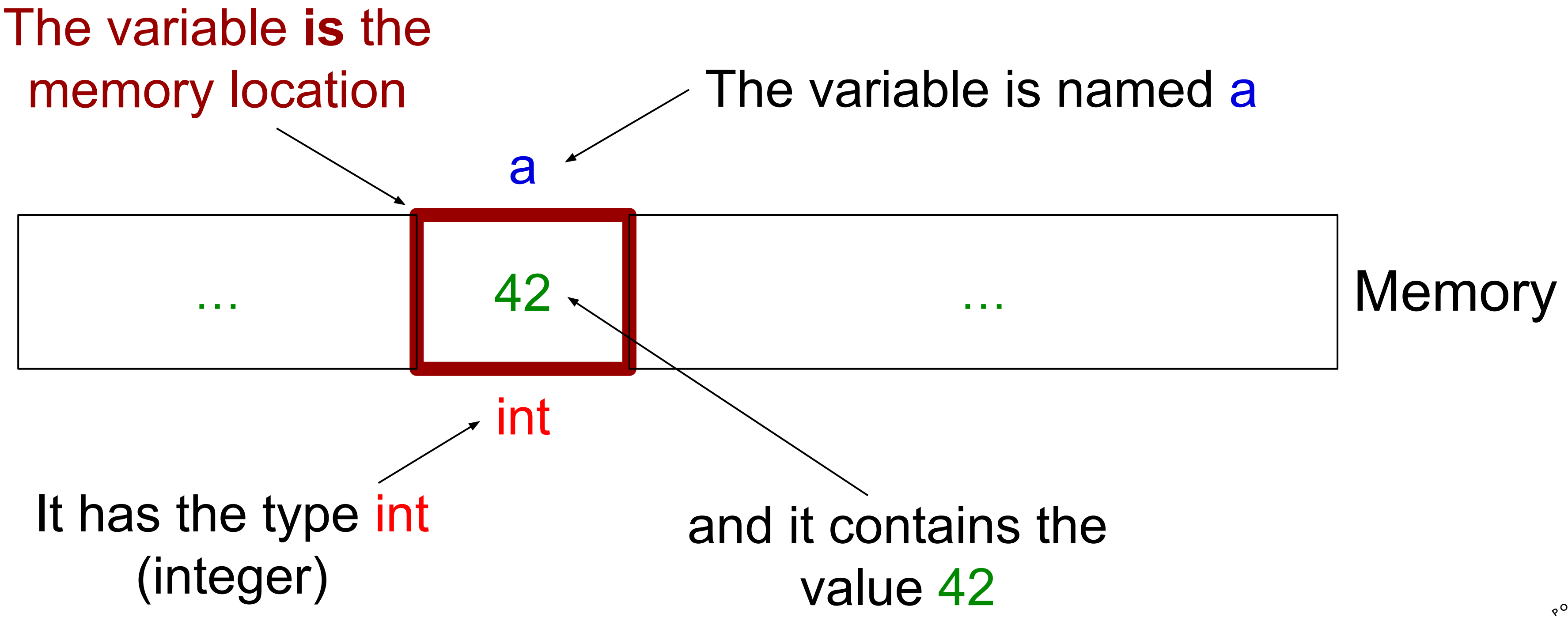
- Variable declaration: `type var;`

- Common types:
  - Integer: `char, short, int, long, long long,`
  - Real numbers: `float, double`
  - String: `char*` (not really, but enough for the moment)
  - Array: `type[]`

- Type conversion
  - Implicit cast when no information is lost
  - Explicit with a cast operator otherwise: `(type)`

Variables in C

- A variable **is a memory location** that has
  - A name: the symbol that identifies the memory location
  - A type: the nature of the memory location
  - A value: the content of the memory location



The variable **is** the memory location
The variable is named a
a
…
42
…
Memory
int
It has the type int (integer)
and it contains the value 42

Common types in C

- Integer numbers:
  - char (1 byte)
  - short (2 bytes)
  - int (implementation specific, most of the time 4 bytes)
  - long (4 bytes)
  - long long (8 bytes)
  - prefix with unsigned for an unsigned integer, otherwise signed
- Real numbers:
  - float (4 bytes)
  - double (8 bytes)
- String
  - char* (implementation specific)
    (Note: char* is not a string at all, but as a first approximation, imagine that it's the case)
- Array (a sequence of elements with the same type)
  - type[] (for example int[] for an array of int)

The pseudo-type void

- void is a pseudo-type used to indicate that a function returns nothing

```
void say_hello() {
  printf("hello\n");
  return; // optional
}
```

The literals in C

- Integer: an integer value such as `0`
  - Encoded as an int (4 bytes)
  - If suffixed with ‘l’, encoded as a long long (8 bytes), e.g., `0l`
- Character: a letter surrounded by a single quote such as `'a'`
  - A character is converted into an integer named its ascii code
  - And encoded as a char (1 byte)
  - That’s why the type char in C is considered as an integer type
- Real number: a number with a dot such as `3.14`

  (you can also write it as 2.13e-2, which means $2.13*10^{-2}$)
  - Encoded as a float (4 bytes)
  - If suffixed with l, encoded as a double (8 bytes), e.g., `3.14l`
- String: a sequence of characters surrounded by a double quote, such as `"Hello, world!\n"`

Declaring a variable

- Each variable in C has to be explicitly declared
  - With `type name;`
  - The type of a variable is fixed and cannot change

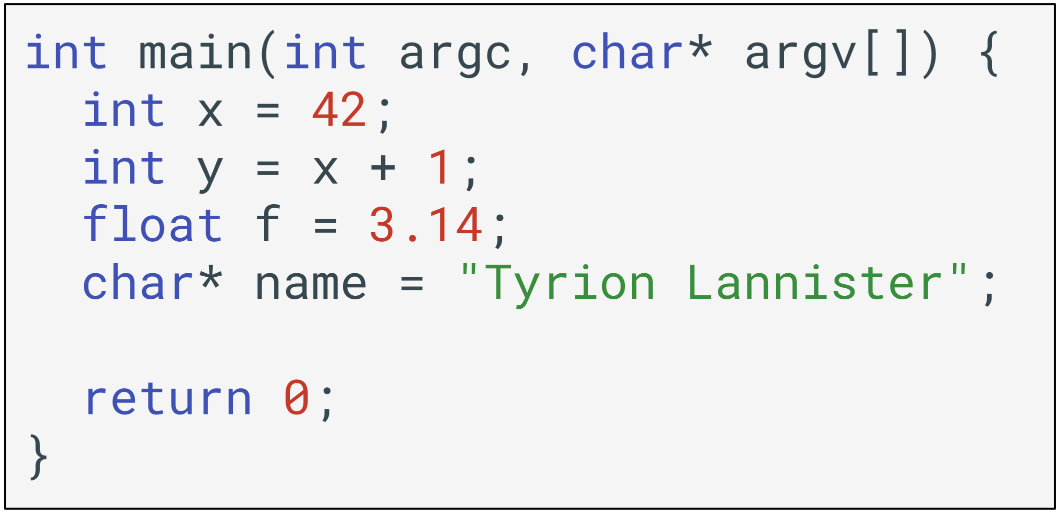
```
int main(int argc, char* argv[]) {
  int x;         /* declare an int */
  float f;       /* declare a float */
  char* name;    /* declare a string */
  int tab[4];    /* declare an array of 4 int */

  x = 42;
  f = 3.14;
  name = "Tyrion Lannister";
  tab[0] = 42; /* set the first elements of the array */

  return 0;
}
```

Declaring a variable

- You can also declare a variable and gives it an initial value in a single statement



```
int main(int argc, char* argv[]) {
  int x = 42;
  int y = x + 1;
  float f = 3.14;
  char* name = "Tyrion Lannister";

  return 0;
}
```

Declaring a variable

- Or declare multiple variables in a single statement

```cpp
int main(int argc, char* argv[]) {
  int x, y = 3, z;

  return 0;
}
```

Constant

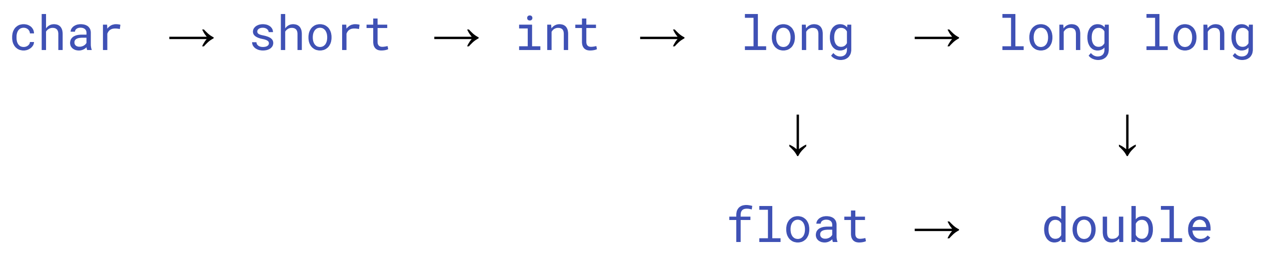
- A variable can be declared constant with the `const` keyword
  - Assign a value when it is declared
  - Cannot change later

- Avoid bugs (read-only variable) and enables optimizations

```
int main(int argc, char* argv[]) {
  const int x = 42;
  printf("%d\n", x);
  //x = 33; => forbidden
  return 0;
}
```

Type conversion and cast operator

- You can convert a value from a type s to a type d with a cast
  - Implicit cast when no information is lost



char → short → int → long → long long

long ↓ float

long long ↓ double

float → double

  - Explicit cast with a cast operator otherwise: (type)

```
char   a = 'a';         // 'a' => 97
int    b = a;           // 97
float  c = b;           // 97.0
double d = c;           // 97.0

short  e = (short)d;    // 97
char   f = (char)97.3;  // 97 => 'a'
```

Printing a variable

- The `printf` function prints its arguments on the terminal
  - Take as argument a format followed by arguments
  - Note: an integer smaller than 4 bytes is promoted to 4 bytes

	4 bytes	8 bytes	Other
signed decimal	%d	%ld	
unsigned decimal	%u	%lu	
hexadecimal	%x	%lx	
character	%c		
string			%s

```
printf("Bip: %i %f %c %s\n", 42, 3.14, 'a', "bap");
=> “Bip: 42 3.14 a bap”
```

Comparison with python

- C is an explicitly typed language
  - You have to explicitly declare a variable
  - And gives it a type at declaration
  - And the type cannot change later

```
int x;
x = 42;
// x = "hello" => error
```

- Python is a dynamically typed language
  - A variable is automatically created when it is used
  - Its type is dynamically deduced from the assigned value
  - The type can change dynamically

```
x = 42
# the type of x can change dynamically
x = "hello"
```

Pro and cons

- Explicit typing
  - + Detect typing bugs at compilation
  - + Simplify memory management since the size of a variable is known at compilation time
  - - More work for the developer

- Dynamic typing
  - - Detect typing bugs too late, at runtime!
  - - Complexify memory management since the size of a variable can change during execution (⇒ performance overheads)
  - + Less work for the developer

Key concepts

- Variable declaration: `type var;`
- Common types:
  - Integer: `char, short, int, long, long long,`
  - Real numbers: `float, double`
  - String: `char*` (not really, but enough for the moment)
  - Array: `type[]`
- Type conversion
  - Implicit cast when no information is lost
  - Explicit with a cast operator otherwise: `(type)`