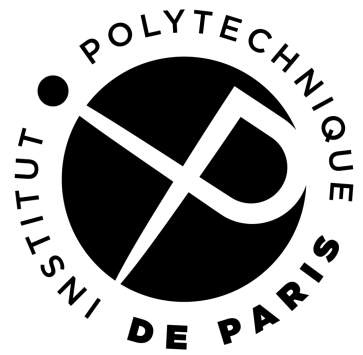

Arrays and structures

Bachelor of Science - École polytechnique

gael.thomas@inria.fr

Key concepts

- Array: homogenous
  - To declare an array: `type var[n];`
  - To access an array: `var[n]`

- Structure: heterogeneous
  - To define a type: `struct name_of_the_struct { … };`
  - To declare a structure: `struct name_of_the_struct var;`
  - To access a field: `var.field_name`

Complex data structures

- The C language provides two kinds of complex data structures
  - Array: to store homogenous elements
  - Structure: to store heterogeneous elements

An array of 6 `int`

A structure with a `char`, an `int` and a `float`

Array declaration

- To declare an array: `type name[n]`
  - `type` gives the type of the elements
  - `name` is the name of the variable
  - `[...]` indicates that we declare an array
  - `n` is the number of elements of the array

```
int main(int argc, char** argv) {
  int tab[3];
  return 0;
}
```

Note: the number of elements is fixed at declaration and cannot change later

Declaration and initialization

- We can initialize an array when we declare it
  - With `type name[n] = { v0, v1, v2, ... };`
  - In this case, we can omit `n` (deduced from the initializer)

```
int main(int argc, char** argv) {
  int tab[] = { 1, 2, 3 };
  return 0;
}
```

Array access

- To access an array: `name[idx]`
  - `name` gives the name of the variable
  - `[...]` indicates an array access
  - `idx` gives the index of the accessed element

```
int main(int argc, char** argv) {
  int tab[3];
  for(int i=0; i<3; i++) {
    tab[i] = i + 1;
  }
  return 0;
}
```

Array and function call

- To declare a function parameter with an array type
  - `type name[]`

```
void f(int tab[]) {
 for(int i=0; i<3; i++) {
    printf("%d\n", tab[i]);
  } // print 1, 2, 3
}

int main(int argc, char** argv) {
  int tab[3] = { 1, 2, 3 };
  f(tab);
  return 0;
}
```

Array and function call

- To declare a function parameter with an array type
  - `type name[]`

```
void f(int tab[]) {
 for(int i=0; i<3; i++) {
    printf("%d\n", tab[i]);
  } // print 1, 2, 3
}

int main(int argc, char** argv) {
  int tab[3] = { 1, 2, 3 };
  f(tab);
```

Note: if `f` modifies `tab`, this modifies the `tab` of `main`
(we say that an array is passed by pointer - explained in a next lesson)

Structure declaration

- To declare a structure: first, define a new type with

```
struct name_of_the_struct {
  type1 field_name_1;
  type2 field_name_2;
  ...
};
```

```
struct strange {
  char c;
  int i;
  float f;
};
```

`c`, `i` and `f` are called the fields of the structure

Structure declaration

- To declare a structure: first, define a new type with

```
struct name_of_the_struct {
  type1 field_name_1;
  type2 field_name_2;
  ...
};
```

```
struct strange {
  char c;
  int i;
  float f;
};
```

- The new type is used to create symbols to access the fields
  - `c` is the 0th element of the structure
  - `i` is the 1st element of the structure
  - `f` is the 2nd element of the structure

Structure declaration

- Then you can declare a variable with the new type

```
struct name_of_the_struct var_name;
```

```
struct strange {
  char c;
  int i;
  float f;
};

int main(int argc, char* argv[]) {
  struct strange var;
  return 0;
}
```

Declaration and initialisation

- You can initialize a structure when you declare it with

  `= { .field1: val1, .field2: val2, ... };`

```
struct strange {
  char c;
  int i;
  float f;
};

int main(int argc, char* argv[]) {
  struct strange var = { .c: 'a', .i: 3, .f: 3.14 };
  return 0;
}
```

Declaration and initialisation

- Or with the fields given in the definition order

```
= { val1, val2, ... };
```

```
struct strange {
  char c;
  int i;
  float f;
};

int main(int argc, char* argv[]) {
  struct strange var = { 'a', 3, 3.14 };
  return 0;
}
```

Structure access

- To access a field, use the symbol dot

  `var_name.field_name`

```c
struct strange {
  char c;
  int i;
  float f;
};

int main(int argc, char* argv[]) {
  struct strange var;
  var.c = 'a';
  var.i = 3;
  var.f = 3.14;
  printf("%c %d %f\n", var.c, var.i, var.f);
  return 0;
}
```

Structure and functions

- To declare a function parameter with a `struct` type

```
struct name_of_the_struct arg_name
```

```
void f(struct strange var) {
  printf("%c %d %f\n", var.c, var.i, var.f);
}

int main(int argc, char* argv[]) {
  struct strange var = { .c: 'a', .i: 3, .f: 3.14 };
  f(var);
  return 0;
}
```

Structure and functions

- To declare a function parameter with a `struct` type

```
struct name_of_the_struct arg_name
```

```
void f(struct strange var) {
  printf("%c %d %f\n", var.c, var.i, var.f);
}

int main(int argc, char* argv[]) {
  struct strange var = { .c: 'a', .i: 3, .f: 3.14 };
  f(var);
  return 0;
```

Note: if `f` modifies `var`, this **does not** modifies the `var` of `main`
(we say that an array is passed by copy - explained in a next lesson)

Key concepts

- Array: homogenous
  - To declare an array: `type var[n];`
  - To access an array: `var[n]`

- Structure: heterogeneous
  - To define a type: `struct name_of_the_struct { … };`
  - To declare a structure: `struct name_of_the_struct var;`
  - To access a field: `var.field_name`