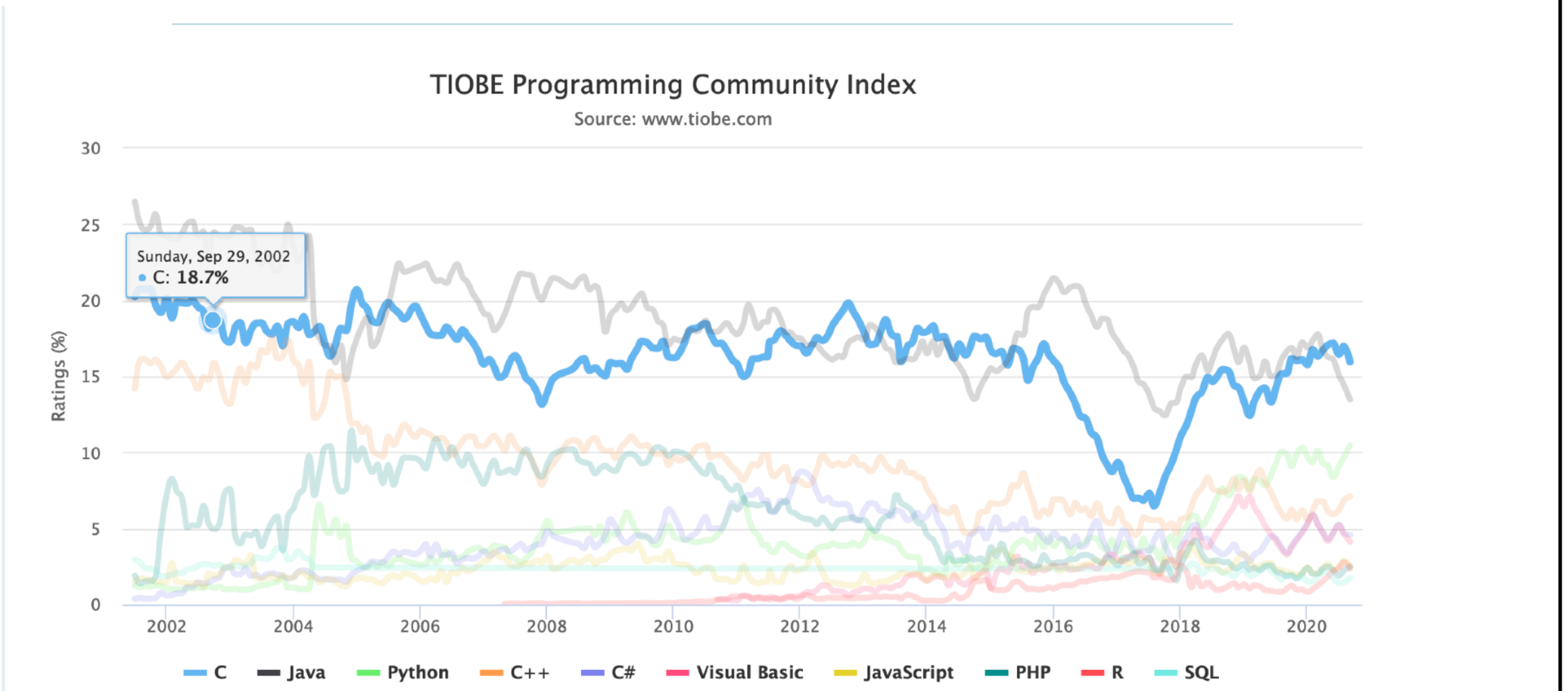
Introduction to C

Olivier Mattelaer
UCLouvain
CP3 & CISM

Why C

Tiobe Ranking



- C is #1 (very lucky this year)
  - ➡ 4 of the 5 top program are C related
  - ➡ Including Python and C++
- C is also useful for Cuda

Program of today

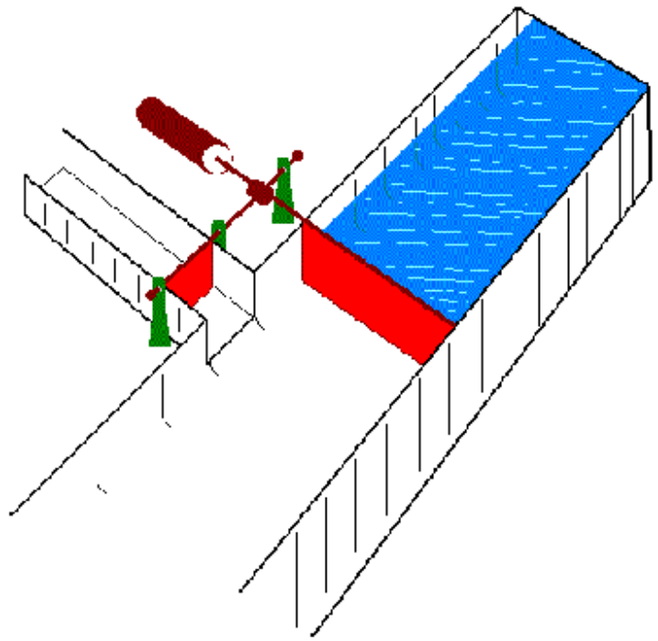
- Basic of C
  - ➡ Type of language
  - ➡ Hello World
  - ➡ Variable
    - ✦ Type of variable
  - ➡ Arrays
  - ➡ Pointers
  - ➡ Functions
  - ➡ Conditional
  - ➡ Data structure
  - ➡ Dynamical memory

This Afternoon

- Basic of C++
  - Introduction to Class/object in C++
  - (Multi) Inheritance

What is a computer?

- A computer has a lot of transistor
  - ➡ This allows to do logic operation like summing number
  - ➡ Binary operation



Nice video on how to do math with domino like transistor:

https://www.google.com/search?client=firefox-b-d&q=doing+math+with+domino+computer#kpvalbx=_tIVbX-zkCMi1sAevpr_ADQ26

Do we need to know that?

- NO !!
  - ➡ We are not re-inventing the wheel
    - ✦ We will NOT use binary number
    - ✦ We will NOT code what the transistor need to do
  - ➡ We will write a task for the computer and use the work of someone else to convert such task into operation on the binary representation
    - ✦ We will use abstraction

C language

C (programming language)

From Wikipedia, the free encyclopedia

"C Programming Language" redirects here. For the book, see The C Programming Language.

C (/siː/, as in the letter *c*) is a general-purpose, procedural computer programming language supporting structured programming, lexical variable scope, and recursion, with a static type system. By design, C provides constructs that map efficiently to typical machine instructions. It has found lasting use in applications previously coded in assembly language.

- C is quite low level language
  - ➡ Allows to generate very efficient machine code
    - ✦ Efficiency of the code depends of the language but also of the algorithm

Hello World

```
1  // my first program in C
2  #include <stdio.h>
3
4  int main()
5  {
6    printf("Hello World!\n");
7  }
```

http://cpp.sh/3okrv

- line1: Comment
  - ➡ also /* … */
- line 2: preprocessor directive:
  - ➡ Include a section of standard C code in the code
- line 3: empty line: do nothing (but clarity for human reader)
- line 4: declaration of a function
  - ➡ main is a special function which is run automatically
  - ➡ starts and stops with the braces (line 5 and 7)
- Statement. Send character to the output device
  - ➡ Note the **semi-column** at the end of the line

Compile the code

C++

Simplest command

Make FILENAME_NO_EXT

- Make is NOT a compiler but a program that knows how to compile
  - ➡ No extension to FILENAME

Calling the compiler:

cc -o EXECNAME input.c

- Convention to call c code with .c

Problem

https://ideone.com/

Select C (bottom left)

http://www.cpp.sh/2dd

https://www.tutorialspoint.com/compile_c_online.php

Run the code

./EXECNAME

Simple code print multiplication table

http://cpp.sh/4odwq

```c
// my first program in C
#include <stdio.h>

int main()
{

  printf("Multiplication table of 5:\n");
  printf(" 5 * 1 = 5 \n");
  printf(" 5 * 2 = 10 \n");
  printf(" 5 * 3 = 15 \n");
  printf(" 5 * 4 = 20 \n");
  printf(" 5 * 5 = 25 \n");
  printf(" 5 * 6 = 30 \n");
  printf(" 5 * 7 = 35 \n");
  printf(" 5 * 8 = 40 \n");
  printf(" 5 * 9 = 45 \n");
  printf(" 5 * 10 = 50 \n");
}
```

- What's wrong with this code?
  - ➡ Maintainability

Variable

http://cpp.sh/522d2

```c
// my first program in C
#include <stdio.h>

int main()
{

  int i = 5;
  printf("Multiplication table of %d:\n", i);
  printf(" %d * 1 = %d \n", i, i);
  printf(" %d * 2 = %d \n", i, 2*i);
  printf(" %d * 3 = %d \n", i, 3*i);
  printf(" %d * 4 = %d \n", i, 4*i);
  printf(" %d * 5 = %d \n", i, 5*i);
  printf(" %d * 6 = %d \n", i, 6*i);
  printf(" %d * 7 = %d \n", i, 7*i);
  printf(" %d * 8 = %d \n", i, 8*i);
  printf(" %d * 9 = %d \n", i, 9*i);
  printf(" %d * 10 = %d \n", i, 10*i);
}
```

- Make "5" a parameter
  - ➡ Abstract the code for any value

```c
int i = 5;
```

- Note that
  - I say that this is an integer
  - That it's (initial) value is 5

While loop

```
1  // my first program in C
2  #include <stdio.h>
3
4  int main()
5  {
6
7    int i = 5 ;
8    printf("Multiplication table of %d:\n", i);
9    int j=1;
10   while(j<11){
11         printf(" %d * %d = %d \n", i,j, i*j);
12         j = j +1;
13
14   }
15
16 }
```

- Spaces are not important (line9)
  - ➡ "=" is the assignment operation not a mathematical operation
  - ➡ "j" will change value while looping (line 10-14)

For loop

cpp.sh/75vpk

```c
// my first program in C
#include <stdio.h>

int main()
{

  int i = 5.;
  printf("Multiplication table of %d:\n", i);
  for (int j=0; j<10; j++){
            printf(" %d * %d = %d \n", i,j+1, i*(j+1));
  }

}
```

- j++: means “add one to the value of j”
- Quite common to count from 0 in C

Loop

- For (int i=0; i< …; i++) {}
- while(condition) {code}
- Do{ code }while(condition);

Loop special keyword

- continue
  - Go to the next step in loop (bypass any following lines in the loop for this step)
- break
  - Stop the loop (resume main code)

Variable

```c
// my first program in C
#include <stdio.h>

int main()
{

  int i = 5;
  float x=1.0;
  double c =1.0;
  char a = 'h';
```

- No type for string
  - ➡ But wait for it
- Boolean supported since 99
  - ➡ Requires “#include stdbool.h”

```c
printf("How to print: %d %c %f %f:\n", i,a,x,c);
```

- Note you can not define twice the same variable name
- Variable name have a “scope”, only available locally

Functions

```c
// my first program in C
#include <stdio.h>

void print_table(int tableof, int maxmul){

  for(int j=1; j<(maxmul+1); j++){
      printf("the product of %d and %d is %d\n", tableof, j, tableof*j);
      }

}


int main()
{
 print_table(4,10);
 print_table(5,11);

}
```

- Function allows to reuse a piece of code with argument
- Other variable are not passed to the function
  - ➡ You can define a variable with the same name in both block. They will not conflict and not share the value
- Argument can not be alter

What if I want to change a variable via a function?

- That's where the address/pointer are useful

cpp.sh/2pqel

Address

```c
int i = 5;
```

- A variable contains a value
  - ➡ That value can change with time
  - ➡ That value is store on RAM at a given place
  - ➡ This place is called the “address” of the variable

http://cpp.sh/932uo

```c
// my first program in C
#include <stdio.h>

int main()
{

  int i = 5;
  printf("Multiplication table of %d:\n", i);
  printf("i is store in ram at adress %p", &i);
}
```

```
Multiplication table of 5:
i is store in ram at adress 0x7078f5bfb7bc
```

Address

```c
int i = 5;
```

- A variable contains a value
  - ➡ That value can change with time
  - ➡ That value is store on RAM at a given place
  - ➡ This place is called the "address" of the variable
- Seems a useless concept
  - ➡ The place in RAM is not predictable
- Many C function will take as argument the address and not the variable itself
  - ➡ So this is actually often use in
    - ✦ C/C++/Python/…

Can I store the address in a variable?

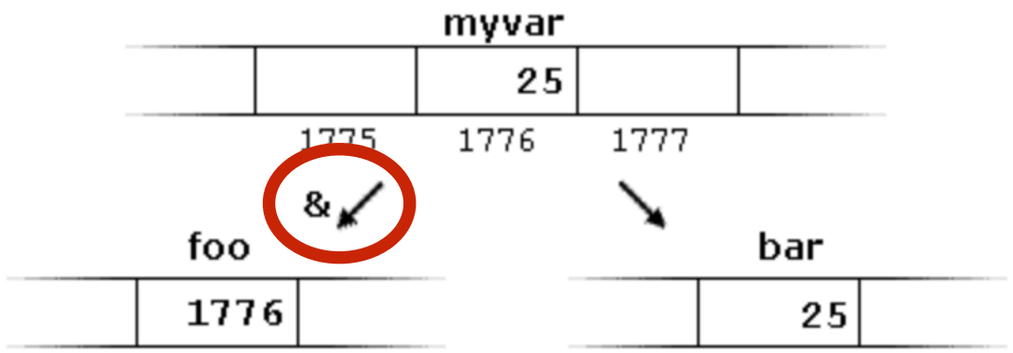
- Yes you can store the address.
  - ➡ As C is strongly typed, you have a type for that

```
int* pi = &i;
printf("i=%d is store in ram at adress %p\n", i, pi);
```

  - ➡ Each type of numbers have various size (number of bit) in memory, so we have a type of address for any type of value.
  - ➡ This is call pointer.
  - ➡ “easy syntax”: add a “*” to the name of the original type
    - ✦ float*
    - ✦ bool*
    - ✦ char*
  - ➡ Possible to get the value associate to a pointer:
    - ✦ *pi

Basic of pointer

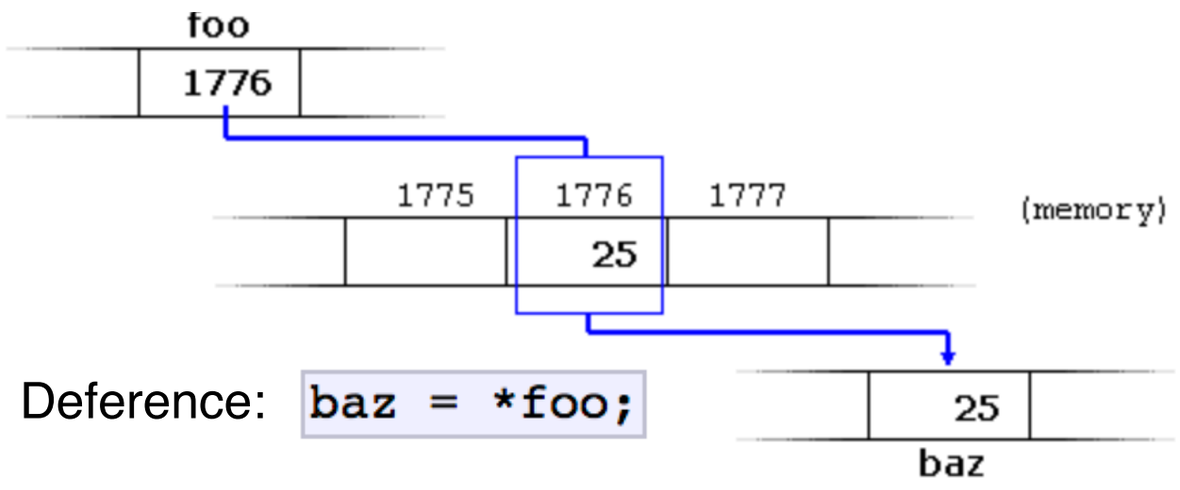
Pointer = position in memory of the variable




```
foo = &myvar;
```

```
bar = myvar
```

```
foo = &myvar;
```



Deference: `baz = *foo;`

- Due to deference pointer also have typed:
  - ➡ Those are the type of the variable suffix by a star

```
1 int * number;
2 char * character;
3 double * decimals;
```

What if I want to change a variable via a function?

- That's where the address/pointer are useful

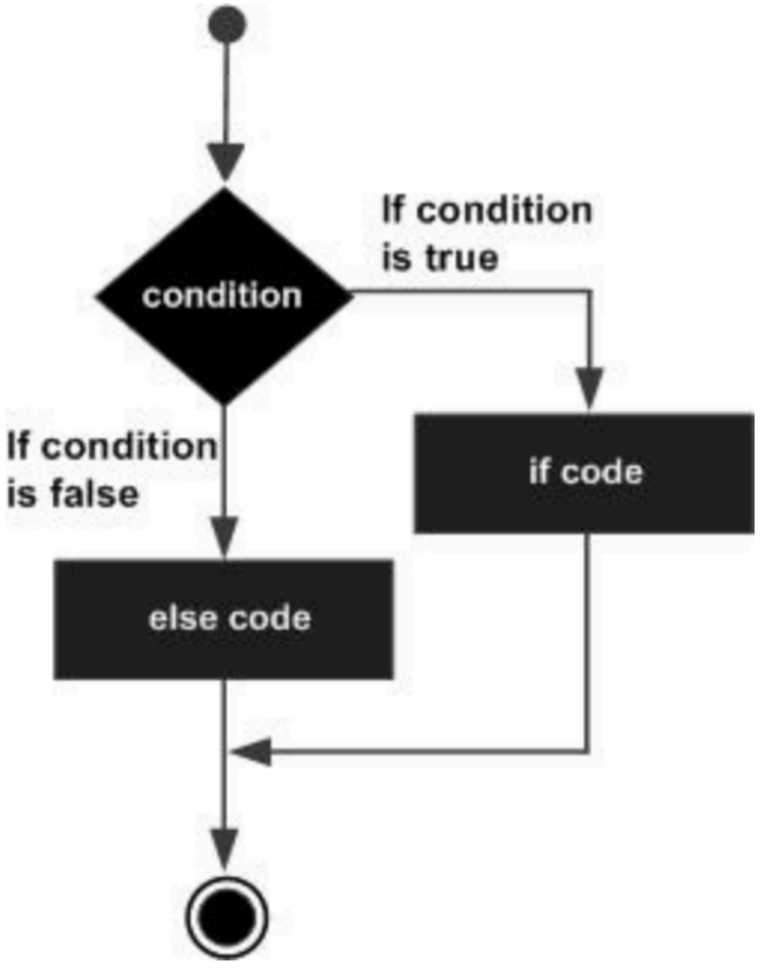
```c
/* function declaration */
void swap(int* x, int* py);

int main () {

   /* local variable definition */
   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );

   swap(&a, &b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}

/* function definition to swap the values */
void swap(int* px, int* py) {

   int temp;
   temp = *px;    /* save the value at address px */
   *px = *py;      /* put the value from adress py into
   *py = temp;    /* put temp into adress py */

   return;
}
```

cpp.sh/2pqel

- You can modify what is store at a given memory location

- So you pass the address and modify the value store at that address

If statement



- Checking condition and react accordingly is the core of programming

```c
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 100;

   /* check the boolean condition */
   if( a < 20 ) {
      /* if condition is true then print the following */
      printf("a is less than 20\n" );
   } else {
      /* if condition is false then print the following */
      printf("a is not less than 20\n" );
   }

   printf("value of a is : %d\n", a);

   return 0;
}
```

- One liner:

```c
int x = (a>0 ? 2 : 4);
printf("x= %d\n", x);
```

https://www.tutorialspoint.com/compile_c_online.php

and/or operation

- Combining condition is of curse crucial

Operator	Meaning
&&	AND
||	OR
!	NOT

```c
if ( a && b ) {
   printf("Line 1 - Condition is true\n" );
}

if ( a || b ) {
   printf("Line 2 - Condition is true\n" );
}

/* lets change the value of  a and b */
a = 0;
b = 10;

if ( a && b ) {
   printf("Line 3 - Condition is true\n" );
} else {
   printf("Line 3 - Condition is not true\n" );
}

if ( !(a && b) ) {
   printf("Line 4 - Condition is true\n" );
}
```

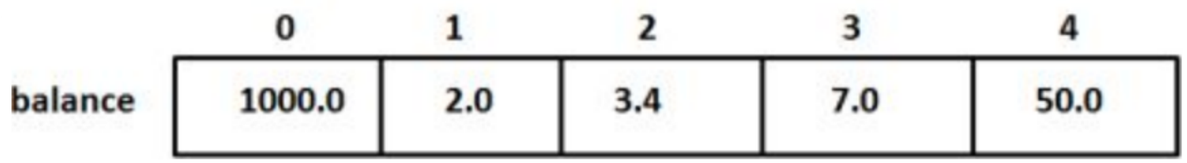
https://www.tutorialspoint.com/compile_c_online.php

Array

- Let's represent a list of number
- The size of an array is fixed!

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

```
balance[4] = 50.0;
```




```c
#include <stdio.h>

int main () {

   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;

   /* initialize elements of array n to 0 */
   for ( i = 0; i < 10; i++ ) {
      n[ i ] = i + 100; /* set element at location i to i + 100 */
   }

   /* output each array element's value */
   for (j = 0; j < 10; j++ ) {
      printf("Element[%d] = %d\n", j, n[j] );
   }

   return 0;
}
```

https://www.tutorialspoint.com/compile_c_online.php

Array and function

- Array are actually pointers…
  - ➡ Those two codes are identical

http://tpcg.io/h9ymMaep

```c
float average(int* myarray, int size){

    float average;
    for (int i =0; i<size; i++){
        printf("Element[%d] = %d\n", i, myarray[i] );
        average += myarray[i];
    }
    average /= size;
    return average;
}
```

```c
float average(int myarray[], int size){

    float average;
    for (int i =0; i<size; i++){
        printf("Element[%d] = %d\n", i, myarray[i] );
        average += myarray[i];
    }
    average /= size;
    return average;
}
```

- You can pass a sub-array to a function

```c
printf( "average  from index 5 is %f\n", average(&n[5], 5));
```

Strings

- No native “strings” type
- You can use an array of char

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

```
char greeting[] = "Hello";
```

- A series of functions simplify handling of strings
  - ➡ Via “include <string.h>”

1	**strcpy(s1, s2);** Copies string s2 into string s1.
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.
3	**strlen(s1);** Returns the length of string s1.

Data structure

- Can we have a special data-type with metadata
  - ➡ Like a "formation"
    - ✦ With the number of student
    - ✦ The name of the formation
    - ✦ The name of the teacher

```c
struct Formation {
    char  title[50];
    char  speaker[50];
    int   nb_student;
};
```

```c
int main( ) {

    struct Formation Lect_C;
    struct Formation Lect_Cpp;

    /* Formation C  initialization*/
    strcpy( Lect_C.title, "C Programming");
    strcpy( Lect_C.speaker,  "O. Mattelaer");
    Lect_C.nb_student = 10;



    /* print Book1 info */
    printf( " Formation \"%s\" given by \"%s\" has %d student",
                Lect_C.title, Lect_C.speaker, Lect_C.nb_student);
```

http://tpcg.io/umjalDnr

More on Data structure

- Can be passed to functions
- Can have pointer
  - ➡ Can be modified within function
  - ➡ Note special syntax to access attribute from pointer

```c
struct Formation {
   char  title[50];
   char  speaker[50];
   int   nb_student;
};

void print_stat(struct Formation formation){

    /* print Book1 info */
   printf( " Formation \"%s\" given by \"%s\" has %d student\n",
            formation.title, formation.speaker, formation.nb_student);

}

int main( ) {

   struct Formation Lect_C;
   struct Formation Lect_Cpp;

   /* Formation C  initialization*/
   strcpy( Lect_C.title, "C Programming");
   strcpy( Lect_C.speaker,  "O. Mattelaer");
   Lect_C.nb_student = 10;


   print_stat(Lect_C);
   return 0;
}
```

```c
void print_stat(struct Formation* formation){

    /* print Book1 info */
   printf( " Formation \"%s\" given by \"%s\" has %d student\n",
            formation->title, formation->speaker, formation->nb_student);
    formation->nb_student +=1;

}
```

Dynamical memory

- You do not always know at compile time the size of all your array

```
int* vector;
int size = 3;
vector = malloc(size * sizeof(int));
```

Array of arbitrary size!!

http://tpcg.io/RRBXOAkj

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {


    int* vector;
    int size = 3;
    vector = malloc(size * sizeof(int));

    if( vector == NULL ) {
      fprintf(stderr, "Error - unable to allocate required memory\n");
      return 1;
   }
    vector[0] = 1;
    vector[1] = 2;
    vector[2] = 3;

    int i =4;
    if(i<3){
        size +=1;
        vector = realloc( vector, size * sizeof(char) );
        if( vector == NULL ) {
            fprintf(stderr, "Error - unable to allocate required memory\n");
            return 1;
        }
        vector[3] = 4;
    }
    printf("size is %d\n", size);
    for(int j=0; j<size; j++){
        printf("%d ", vector[j]);

    }
    free(vector);

}
```

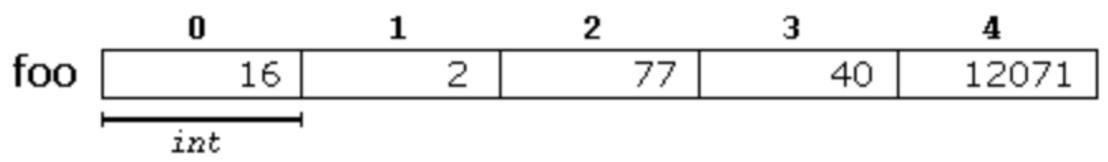
Conclusion

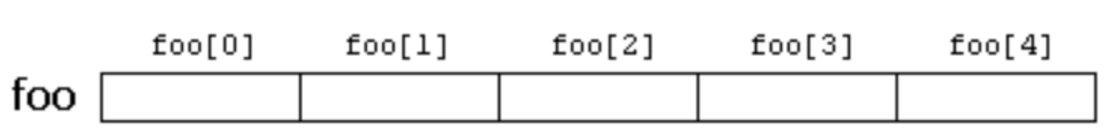
- You need to play with it
  - ➡ Coding is learning by exercise/exploration
  - ➡ Read book on coding style
    - ✦ How to present you code (space/comment/indentation)
    - ✦ Type of good structure/…
- Good understanding of C is key since it defines the basic notion for many language (including Python)
- A lot of this is to learn syntaxes but not only
  - ➡ You need to understand the abstraction

Basic of C++: Array

Array = sequential memory space of the same type

```
int foo [5] = { 16, 2, 77, 40, 12071 };
```






```
1  int foo[] = { 10, 20, 30 };
2  int foo[] { 10, 20, 30 };
```
C++11

- Note the syntax to receive array in a function!
- Array behaves like pointer!

cpp.sh/7aot

cpp.sh/6fzb

```
1  // arrays as parameters
2  #include <iostream>
3  using namespace std;
4
5  void printarray (int arg[], int length) {
6    for (int n=0; n<length; ++n)
7      cout << arg[n] << ' ';
8    cout << '\n';
9  }
10
11 int main ()
12 {
13   int firstarray[] = {5, 10, 15};
14   int secondarray[] = {2, 4, 6, 8, 10};
15   printarray (firstarray,3);
16   printarray (secondarray,5);
17 }
```

Exercise 1

- Check that you can compile the Hello World example
- Define a function that take 3 float and return the average
  - ➡ Explore different method (variable/reference)
- Have Fun
  - ✦ http://www.cplusplus.com/reference
  - ✦ http://www.cplusplus.com/doc/tutorial/

Solution

part I : cpp.sh/6ar2x

part II: cpp.sh/7qfwf

Classes

classes = data structure with functions

data structure = group of data elements grouped together under a single name

- We can define a class "Car"
  - ➡ Defines the structure
    - ✦ Which property available: attribute
      - model, colour, has_autodrive,nb_door
    - ✦ Which function can be applied.
      - change_battery, add_fuel,…
- Class is a new type like "int/float"
  - ➡ Car mytesla;
    - ✦ "mytesla" is an instance of the class CAR

```
1 class Rectangle {
2     int width, height;
3   public:
4     void set_values (int,int);
5     int area (void);
6 } rect;
```

First Example

http://cpp.sh/8ac

```
1  // example: one class, two objects
2  #include <iostream>
3  using namespace std;
4
5  class Rectangle {
6      int width, height;
7    public:
8      void set_values (int,int);
9      int area () {return width*height;}
10 };
11
12 void Rectangle::set_values (int x, int y) {
13   width = x;
14   height = y;
15 }
16
17 int main () {
18   Rectangle rect, rectb;
19   rect.set_values (3,4);
20   rectb.set_values (5,6);
21   cout << "rect area: " << rect.area() << endl;
22   cout << "rectb area: " << rectb.area() << endl;
23   return 0;
24 }
```

- width/height are private
- A public function allows to set those values!
- private attribute ensure that no one mess up those variables.

Visibility of attribute/function

private	protected	public
Only accessible from other instance of the same class Accessible from friends DEFAULT	Accessible from other instance of the same class Accessible from friends Accessible from instance of the derived/child class	Accessible from everywhere where the object is visible READ and WRITE!

```cpp
#include <iostream>
using namespace std;

class Rectangle{
private:
  int width, height;
};

int main(){
  Rectangle A;
  A.width =3;
  A.height=2;
  cout << "width=" << A.width<<endl;
};
```

```
simple.cpp:11:5: error: 'width' is a private member of 'Rectangle'
  A.width =3;
    ^
```

```cpp
#include <iostream>
using namespace std;

class Rectangle{
public:
  int width, height;
};

int main(){
  Rectangle A;
  A.width =3;
  A.height=2;
  cout << "width=" << A.width<<endl
};
```

Constructor

constructor = function called after the object is created

cpp.sh/8lr

```
1  // example: class constructor
2  #include <iostream>
3  using namespace std;
4
5  class Rectangle {
6      int width, height;
7    public:
8      Rectangle (int,int);
9      int area () {return (width*height);}
10 };
11
12 Rectangle::Rectangle (int a, int b) {
13   width = a;
14   height = b;
15 }
16
17 int main () {
18   Rectangle rect (3,4);
19   Rectangle rectb (5,6);
20   cout << "rect area: " << rect.area() << endl;
21   cout << "rectb area: " << rectb.area() << endl;
22   return 0;
23 }
```

- The name of the constructor is the name of the function itself!

- Shortcut for setting attribute

```
Rectangle::Rectangle (int x, int y) : width(x), height(y) { }
```

```
Rectangle::Rectangle (int x, int y) : width(x) { height=y; }
```

Overloading

Overloading = more than one function with the same name

- The name of two functions CAN be the same if the number of argument or the type of argument are different.

```
1  // example: class constructor
2  #include <iostream>
3  using namespace std;
4
5  class Rectangle {
6      int width, height;
7    public:
8      Rectangle (int,int);
9      Rectangle (int l): width(l), height(l){};
10     int area () {return (width*height);}
11 };
12
13 Rectangle::Rectangle (int a, int b) {
14   width = a;
15   height = b;
16 }
17
18 int main () {
19   Rectangle rect (3);
20   Rectangle rectb (5,6);
21   cout << "rect area: " << rect.area() << endl;
22   cout << "rectb area: " << rectb.area() << endl;
23   return 0;
24 }
```

- Any function can be overloaded.
- You can overload basic operation between object like addition:
  - operator +

Overloading

Overloading = more than one function with the same name

Overloadable operators												
+	-	*	/	=	<	>	+=	-=	*=	/=	<<	>>
<<=	>>=	==	!=	<=	>=	++	--	%	&	^	!	|
~	&=	^=	|=	&&	||	%=	[]	()	,	->*	->	new
delete		new[]		delete[]								

cpp.sh/27l

```
1  // overloading operators example
2  #include <iostream>
3  using namespace std;
4
5  class CVector {
6    public:
7      int x,y;
8      CVector () {};
9      CVector (int a,int b) : x(a), y(b) {}
10     CVector operator + (const CVector&);
11 };
12
13 CVector CVector::operator+ (const CVector& param) {
14   CVector temp;
15   temp.x = x + param.x;
16   temp.y = y + param.y;
17   return temp;
18 }
19
20 int main () {
21   CVector foo (3,1);
22   CVector bar (1,2);
23   CVector result;
24   result = foo + bar;
25   cout << result.x << ',' << result.y << '\n';
26   return 0;
27 }
```

Special members

Special members = member functions implicitly defined

Member function	typical form for class C:
Default constructor	`C::C();`
Destructor	`C::~C();`
Copy constructor	`C::C (const C&);`
Copy assignment	`C& operator= (const C&);`
Move constructor	`C::C (C&&);`
Move assignment	`C& operator= (C&&);`

- Default constructor:
  - ➡ Present only if no other constructor exists!
- Destructor ~CLASSNAME:
  - ➡ Perform cleanup (remove dynamical allocated memory) when the object is deleted/out of scope
- Copy Constructor:
  - ➡ Called when you call that class (by value) in a function.
  - ➡ Perform shallow copy of all attribute

```
MyClass::MyClass(const MyClass& x) : a(x.a), b(x.b), c(x.c) {}
```

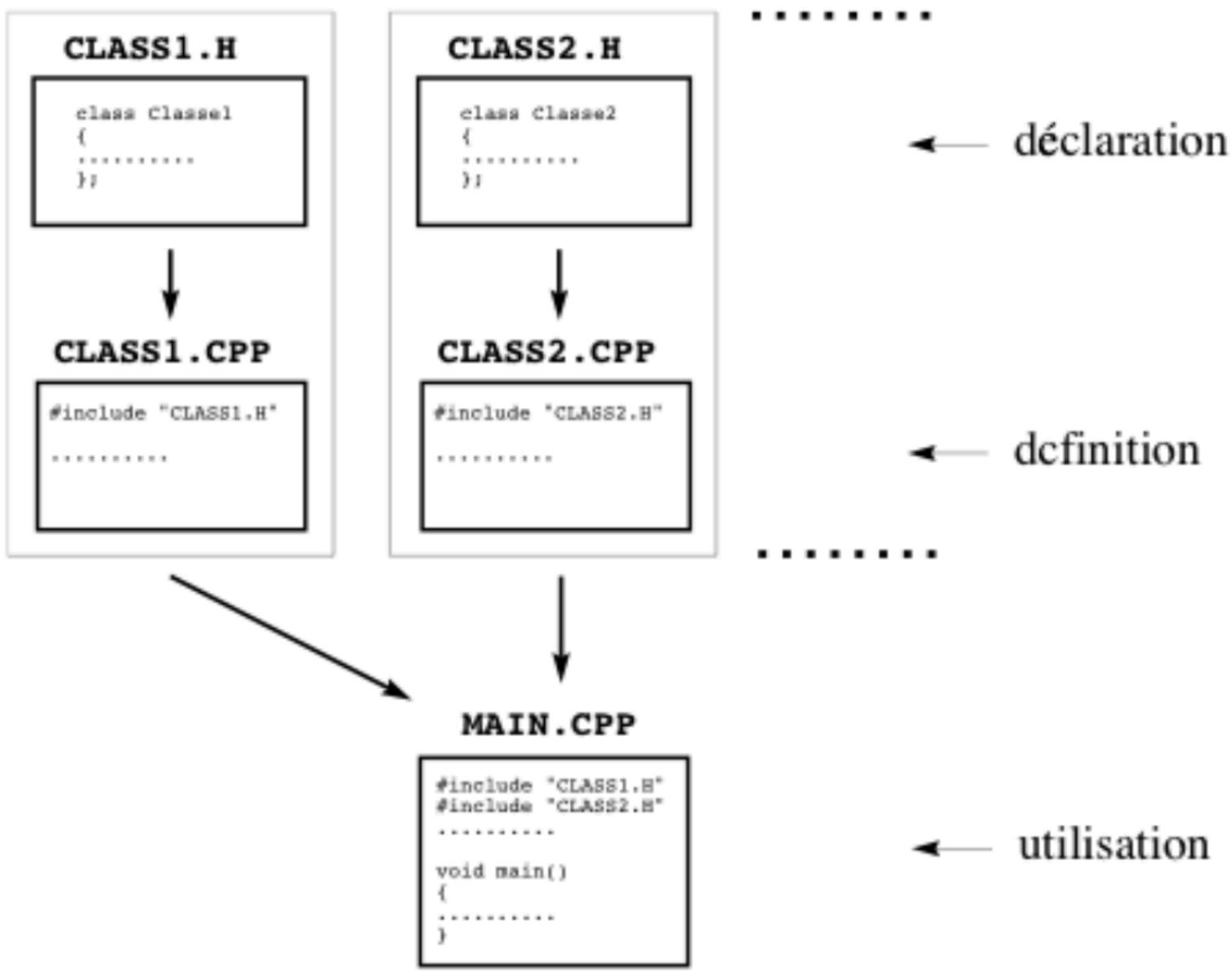
```
1 MyClass fn();             // function returning a MyClass object
2 MyClass foo;              // default constructor
3 MyClass bar = foo;        // copy constructor
4 MyClass baz = fn();       // move constructor
5 foo = bar;                // copy assignment
6 baz = MyClass();          // move assignment
```

cpp.sh/3zgct

Example

```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Rectangle {
    int width, height;
  public:
    Rectangle(){};
    Rectangle (int,int);
    Rectangle (int a, int b, int c): Rectangle(a,b){cout << c<<endl;};
    Rectangle (int l){width=l; height=l;};
    Rectangle(const Rectangle& x){width=x.width; height=x.height; cout<<"copy "<<x.width<<" "<<x.height<<endl;};
    int area () {return (width*height);}
    Rectangle intersection(Rectangle);
};

Rectangle::Rectangle (int a, int b) {
  width = a;
  height = b;
}

Rectangle Rectangle::intersection(Rectangle B){
    //returns a rectangle with the smallest width and height
    Rectangle out;
    if (width < B.width){
        out.width = width;
    }else{
        out.width = B.width;
    };
    if (height < B.height){
        out.height = height;
    }else{
        out.height = B.height;
    };
    return out;
};

int main () {
  Rectangle rect (3);
  Rectangle rectb (2,6,30);
  Rectangle small = rect.intersection(rectb);
  cout << "rect area: " << rect.area() << endl;
  cout << "small area: " << small.area() << endl;
  return 0;
}
```

Code Structure



Exercise II

- Create a class for three dimensional vector
  - ➡ Define function to get/set each component
- Define a function returning the norm(squared) of the vector
  - ➡ x[0]**2+x[1]**2+x[2]**2
- Define the scalar product between two vector:
  - ➡ x[0]*y[0]+ x[1]*y[1]+ x[2]*y[2]
- Define the vectoriel product of two vector

- Define a Class parallelogram
  - ➡ Can be initialised by two vector
  - ➡ Set a function to compute the associated area (norm of vectoriel product)

Solution

cpp.sh/6vgu2c

```cpp
// example: ThreeVector
#include <iostream>
#include <math.h>
using namespace std;

class ThreeVector{
    float v[3];

public:
    ThreeVector(){};
    ThreeVector(float x, float y, float z){ v[0]=x; v[1]=y; v[2]=z;};

    float get_x(){return v[0];};
    float get_y(){return v[1];};
    float get_z(){return v[2];};

    void set_x(float x){v[0] = x;};
    void set_y(float y){v[1] = y;};
    void set_z(float z){v[2] = z;};

    float norm(){return sqrt(v[0]*v[0]+v[1]*v[1]+v[2]*v[2]);};
    float operator * (const ThreeVector& y){return v[0]*y.v[0] + v[1]*y.v[1] +v[2]*y.v[2];}
};

int main () {
    ThreeVector a(1,2,3);
    ThreeVector b(1,0,0);
    cout << "norm a" << a.norm() << endl;
    cout << "norm b" << b.norm() << endl;
    cout << "a*b=" << a*b << endl;
}
```

Solution

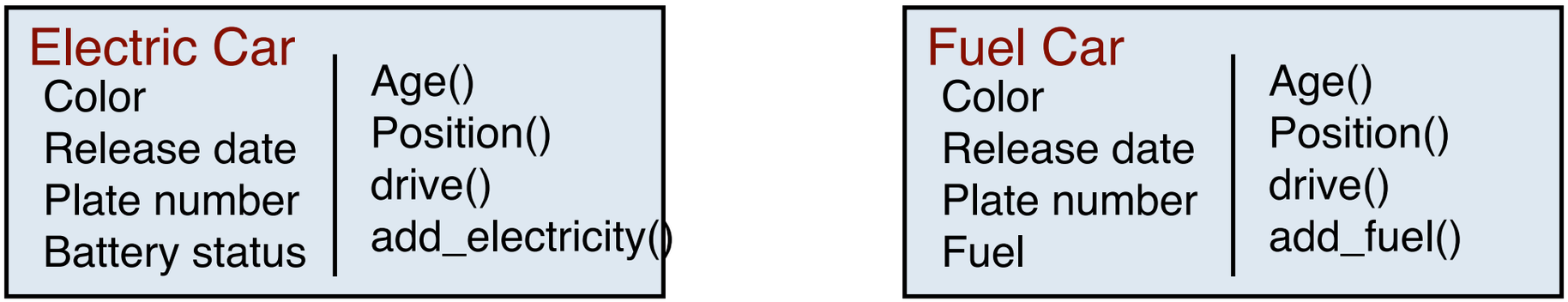
```
class ThreeVector{
    float v[3];

    ThreeVector vmult(ThreeVector);
```

```
    ThreeVector ThreeVector::vmult(ThreeVector second){
        ThreeVector out;
        out.v[0] = v[1]*second.v[2] - v[2]*second.v[1];
        out.v[1] = v[2]*second.v[0] - v[0]*second.v[2];
        out.v[2] = v[0]*second.v[1] - v[1]*second.v[0];
        return out;
    };
```

http://cpp.sh/3pj6pp

```
class Parralelogram{
  ThreeVector first;
  ThreeVector second;
public:
  Parralelogram(ThreeVector f, ThreeVector second): first(f), second(second){};
  float get_area() {return first.vmult(second).norm();}
};


int main () {
    ThreeVector a(1,2,3);
    ThreeVector b(1,0,0);
    cout << "norm a " << a.norm() << endl;
    cout << "norm b " << b.norm() << endl;
    cout << "a*b= " << a*b << endl;
    Parralelogram P(a,b);
    cout << "area of parralelogram " << P.get_area()<<endl;
}
```

Inheritance



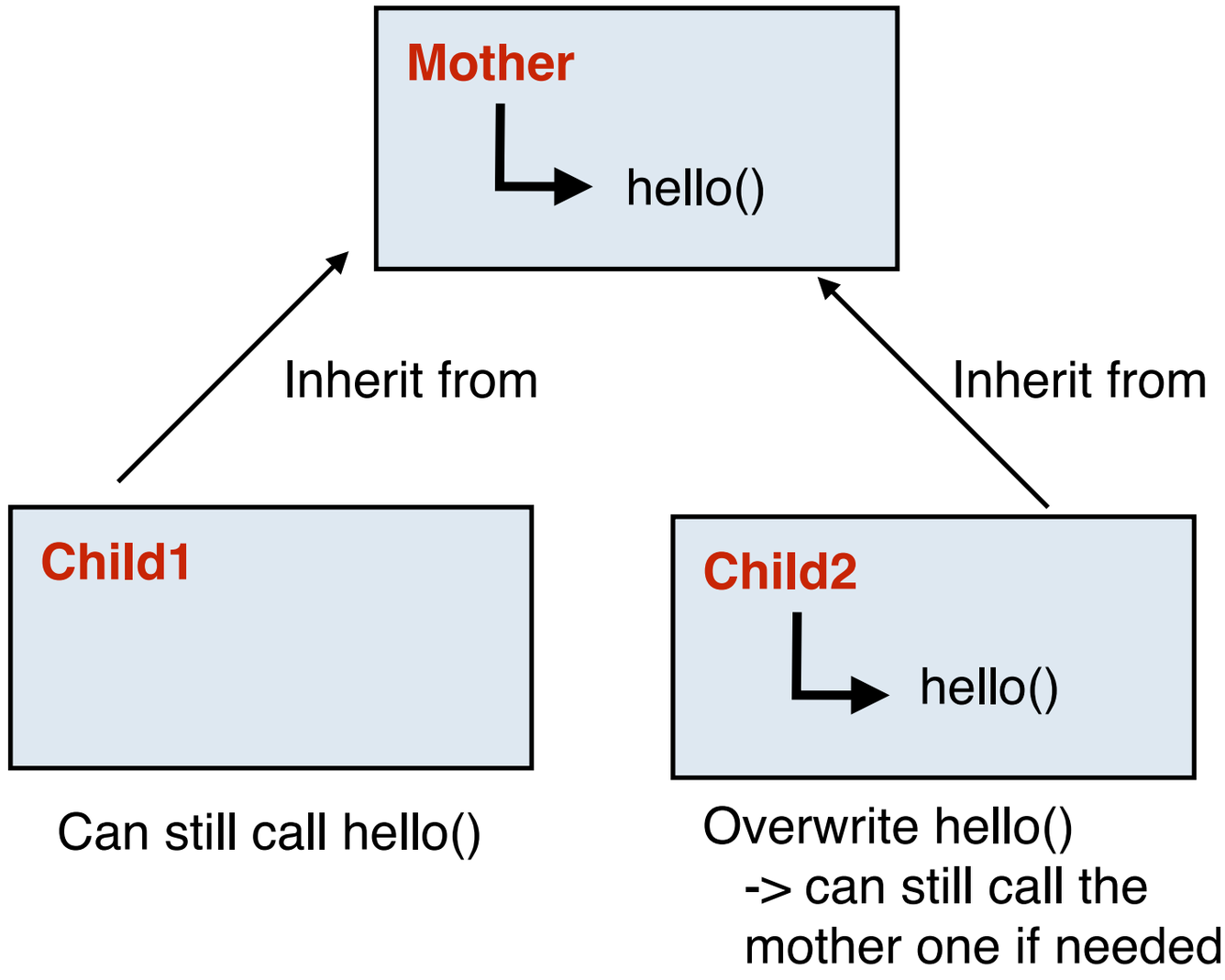
Inheritance

Inheritance = new classes which retain characteristics of the base class.

- The idea is the heritage. What a parent can do, their child can do it too.

cpp.sh/72itc

```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Mother{
public:
    void hello(){
        cout<< "hello from Mother"<<endl;};
};

class Child1: public Mother{};

class Child2: public Mother{

public:
    void hello() {
        Mother::hello();
        cout<< "and from Child2" << endl;};
    };

int main () {
    Child1 test;
    test.hello();

    Child2 test2;
    test2.hello();
}
```



Inheritance

Inheritance = new classes which retain characteristics of the base class.

- The idea is the heritage. What a parent can do, their child can do it too.

cpp.sh/72itc

```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Mother{
public:
    void hello(){
        cout<< "hello from Mother"<<endl;};
};

class Child1: public Mother{};

class Child2: public Mother{

public:
    void hello() {
        Mother::hello();
        cout<< "and from Child2" << endl;};
    };

int main () {
    Child1 test;
    test.hello();

    Child2 test2;
    test2.hello();
}
```

- "public" tells the maximum level of visibility of the attribute coming from the base class
  - Rare case when not set on public
- Private argument are not passed to the child (but they still exits!)
- Constructor/Destructor are **not** passed to the child
- Assignment operator (operator =) are **not** passed to the child

Exercise III

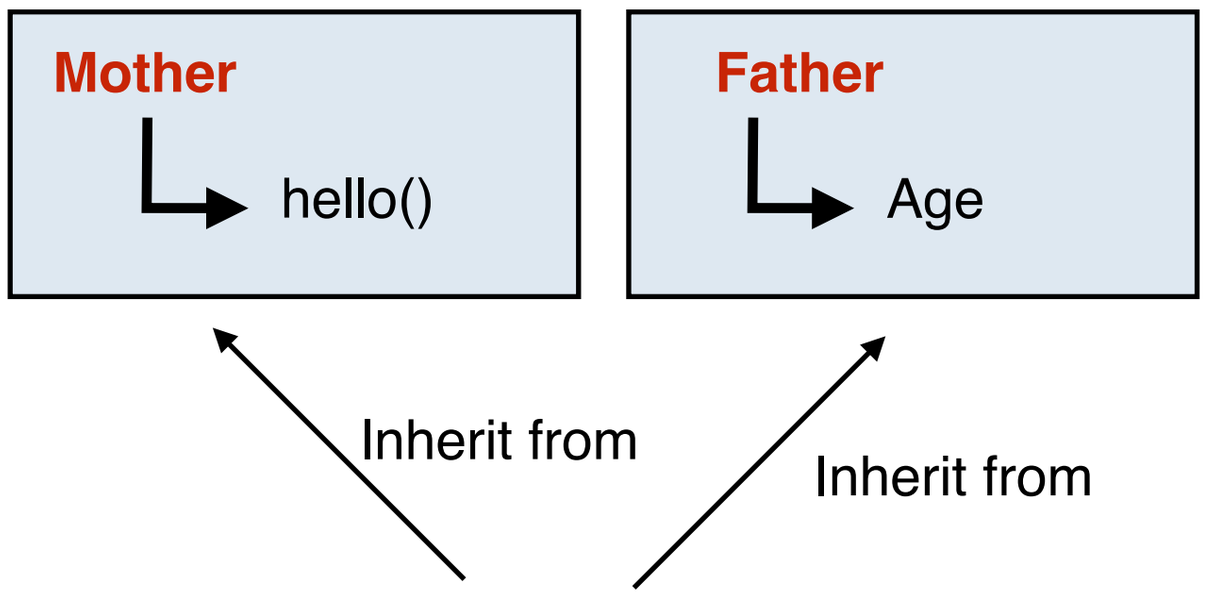
- Create a class Rectangle
- Create a class Square that inherit from class Rectangle
  - ✦ Play with private/public attribute

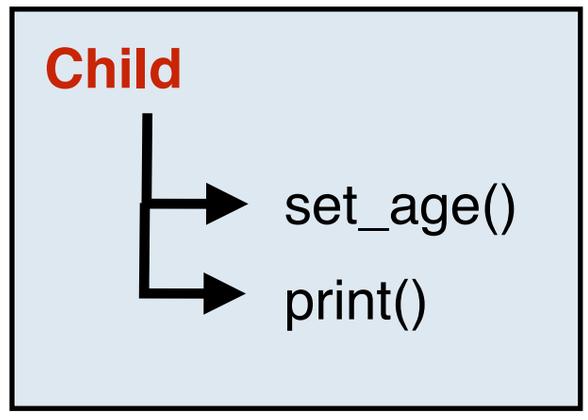
cpp.sh/45lq3

Multi-inheritance

cpp.sh/3nhb

```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Mother{
public:
    void hello(){
        cout<< "hello from Mother"<< endl;};
};

class Father{
protected:
    int age;
public:
    Father(){};
    Father(int x): age(x){};
};


class Child: public Mother, public Father{

public:
    Child(int x){age=x;};

    void print() {hello(); cout<<"my age is " << age;}
    void set_age(int x){age=x;};

};


int main () {
    Child test(3);
    test.hello();
    test.print();
    test.set_age(4);
    test.print();
}
```





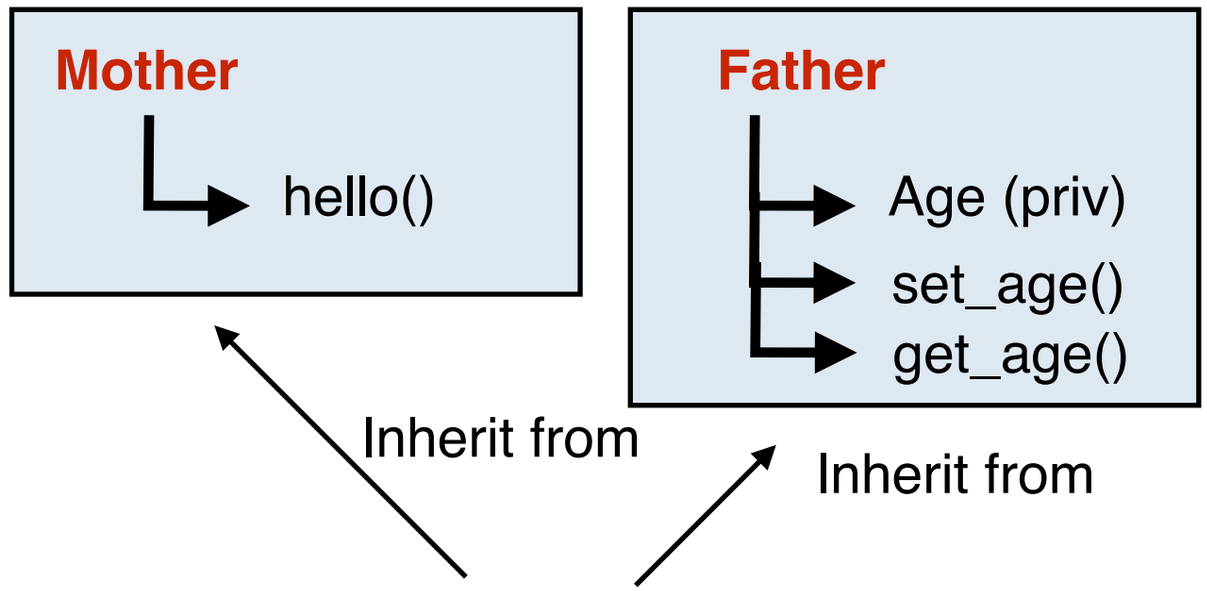
Can still call hello()

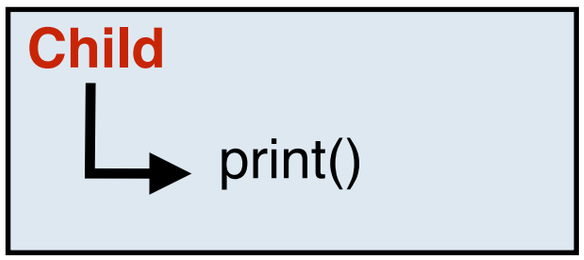
Can access to age (protected)

Multi-inheritance

cpp.sh/8vev

```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Mother{
public:
    void hello(){
        cout<< "hello from Mother"<< endl;};
};

class Father{
    int age;
public:
    Father(){};
    Father(int x): age(x){};
    void set_age(int x){age=x;};
    int get_age(){return age;};
};


class Child: public Mother, public Father{

public:
    Child(int x){set_age(x);};
    void print() {hello(); cout<<"my age is " << get_age();}


};


int main () {
    Child test(3);
    test.hello();
    test.print();
    test.set_age(4);
    test.print();
}
```





Can call hello()

Can not call age (since private)
But can call the public routine of father which set/get the age variable

Exercise III

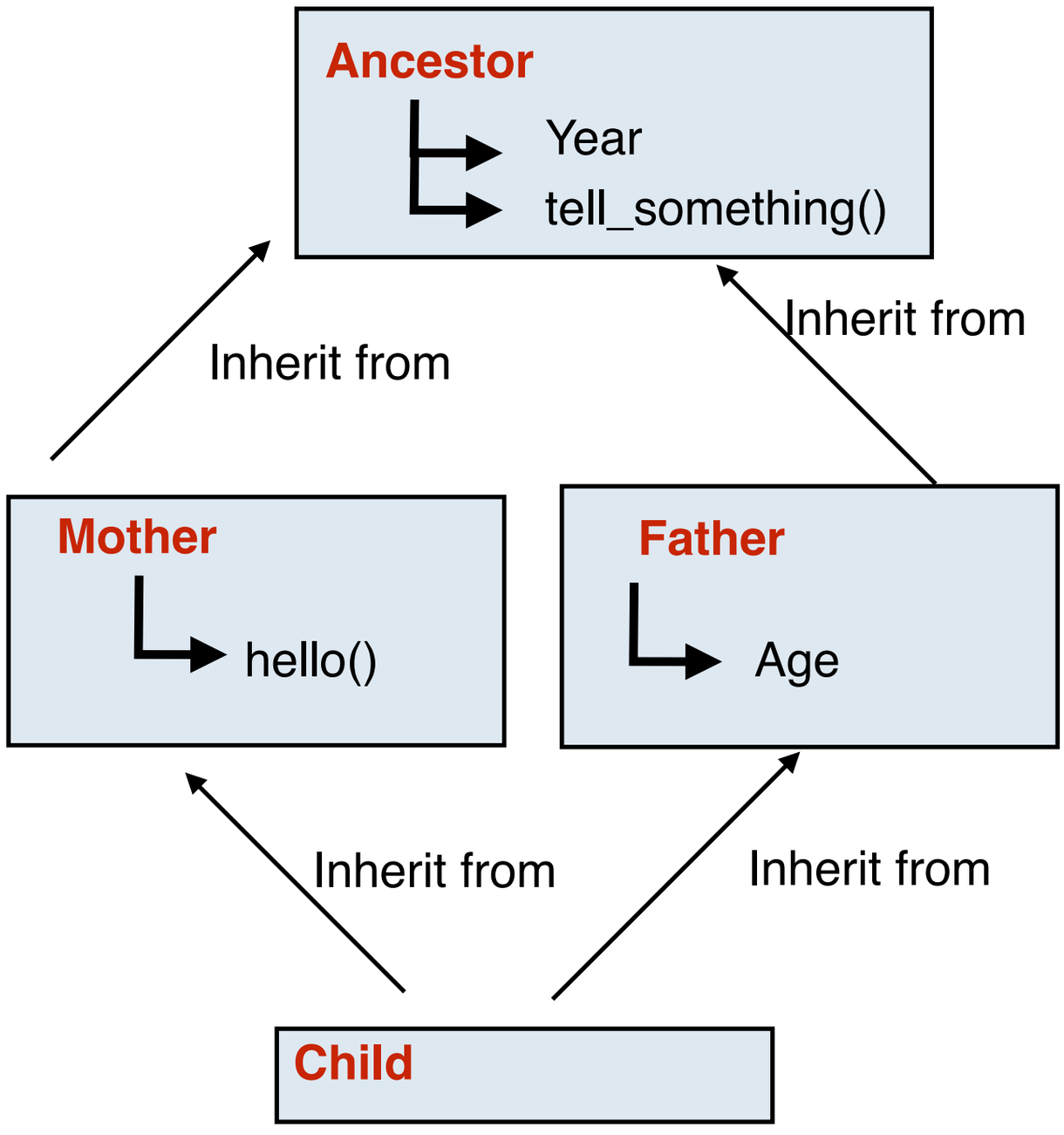
- Update your Rectangle class to have a function returning the smallest Rectangle
- Define a class VectorRectangle
  - Which inherits from your parralelogram class
  - Which inherits from your rectangle class

solution: cpp.sh/2ssxh

Diamond Diagram

cpp.sh/4inoj

```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Ancestor{
public:
    int year;
    void tell_something(){cout<<"In the year "<< year <<endl;};
};

class Mother: public Ancestor{
public:
    void hello(){
        tell_something();
        cout<< "hello from Mother"<< endl;
        };
};

class Father:public Ancestor{
protected:
    int age;
public:
    Father(){};
    Father(int x): age(x){};
};

class Child: public Mother, public Father{
};


int main () {
    Child test;
    test.Mother::year = 1980;
    test.Father::year = 1950;
    test.hello();
    test.Father::tell_something();
}
```



Diamond Diagram

cpp.sh/4inoj

```cpp
// example: class constructor
#include <iostream>
using namespace std;

class Ancestor{
public:
    int year;
    void tell_something(){cout<<"In the year "<< year <<endl;};
};

class Mother: public Ancestor{
public:
    void hello(){
        tell_something();
        cout<< "hello from Mother"<< endl;
        };
};

class Father:public Ancestor{
protected:
    int age;
public:
    Father(){};
    Father(int x): age(x){};
};

class Child: public Mother, public Father{
};


int main () {
    Child test;
    test.Mother::year = 1980;
    test.Father::year = 1950;
    test.hello();
    test.Father::tell_something();
}
```

- Two copy of the Ancestor class
  - ➡ test.Mother::year
  - ➡ test.Father::year
- You can use virtual inheritance to have a single copy
  - ➡ “public virtual Ancestor”
- Consider as bad design in C++
  - ➡ Because C++ sucks on those!

Template

Template = define functions class with generic type

- Repeat yourself is bad but often you have to have the exact same definition but for different type

  ➡ Template is the solution

```cpp
// overloaded functions
#include <iostream>
using namespace std;

int sum (int a, int b)
{
  return a+b;
}

double sum (double a, double b)
{
  return a+b;
}

int main ()
{
  cout << sum (10,20) << '\n';
  cout << sum (1.0,1.5) << '\n';
  return 0;
}
```

cpp.sh/4jq

```cpp
// function template
#include <iostream>
using namespace std;

template <class T>
T sum (T a, T b)
{
  T result;
  result = a + b;
  return result;
}

int main () {
  int i=5, j=6, k;
  double f=2.0, g=0.5, h;
  k=sum<int>(i,j);
  h=sum<double>(f,g);
  cout << k << '\n';
  cout << h << '\n';
  return 0;
}
```

Exercise IV

- Update your four-vector class to include
  - ➡ Scalar Multiplication via Template Method

- Test Multi-Heritage on your class
  - ➡ Test virtual heritage on one/two parent class/…

- Have fun…