DIGITAL ARTS & ENTERTAINMENT

Overview course Programming 1

1. Variables 1
2. Variables 2
3. Variables 3
4. Conditionals
5. Iterations
6. Functions 1
7. Functions 2
8. Arrays
9. Strings – Game
10. Classes 1 – Encapsulation
11. Classes 2 – Static const

howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

DIGITAL ARTS & ENTERTAINMENT

Functions

Part I: the beginning

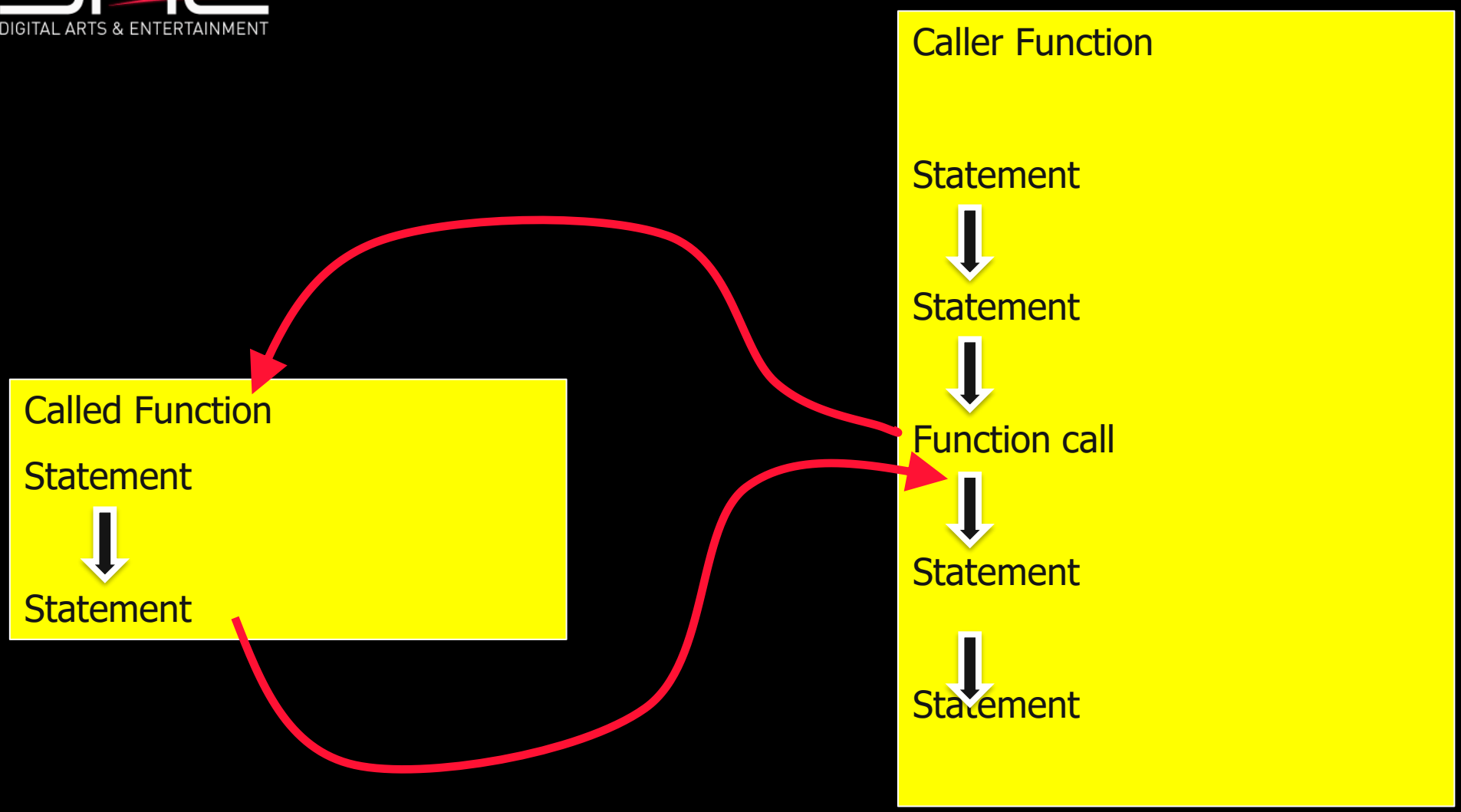
howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

Contents

- How to use functions
- Function value parameters
- Return by value statement
- Function overloading and ambiguity
- Call stack

What?

- Functions are used to split up a *problem* into *sub-problems.*
- *Global* functions*:* a classic function with global scope.
  - Example: the main function.
- *Member* functions: functions with a scope that is limited to the class it is part of. Also called "Methods". See later.

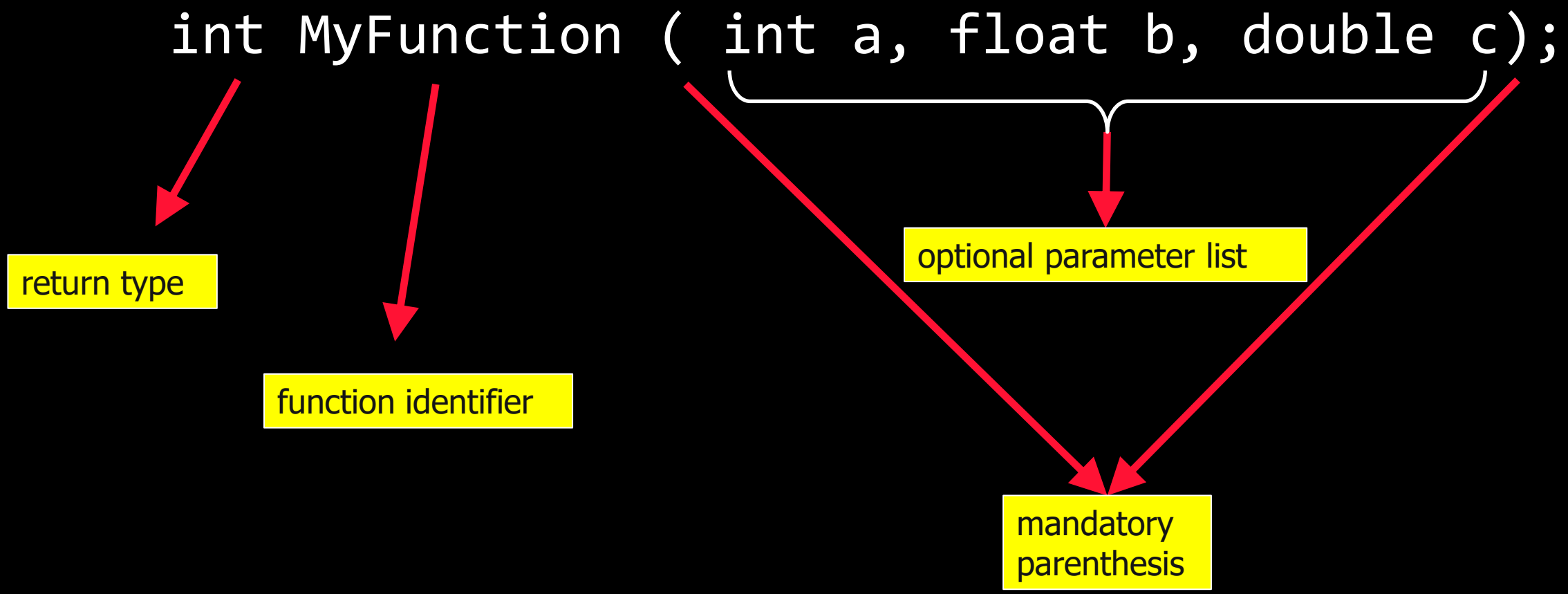

Caller Function
Statement
Statement
Function call
Statement
Statement
Called Function
Statement
Statement

How?

Using functions consists of two steps:

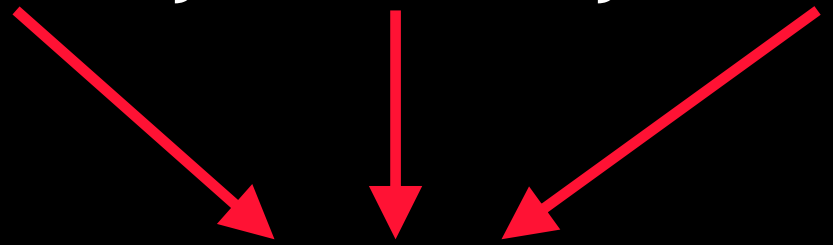
- "Write" the function: what does it do?
- "Call" the function: when does it start?

Function prototype



Function prototype

```
int MyFunction ( int a, float b, double c);
```



parameter types

can be any type, including structs, std::strings, etc...

Function body or definition

```
int MyFunction( int a, float b, double c)
{
    // statements
    return 0;
}
```

Step 1: Forward function declaration

What? The function prototype.

Where? In a console appplication, in the top section of the file, before the start of the main function

```
int CalculateSum(int a, int b);

int main()
{

}
```

Step 1: Forward function declaration

What? The function prototype.

Where? In a framework appplication, in the game.h header file, where the comment tells you to:

```
// Declare your own functions here
void TestFunctions();
int CalculateSum(int a, int b);
```

Step 2: definition "body" of the function

Console: Below the end of the main function

```
int CalculateSum(int a, int b);

int main()
{

}

int CalculateSum(int a, int b)
{
  return a + b;
}
```

Step 2: definition "body" of the function

Framework application: Below the existing definitions, where the comment tells you to:

```
#pragma region ownDefinitions

// Define your own functions here
void TestFunctions()
{
    int result{};
    result = CalculateSum(10, 32);
```

Step 3: calling

Call the function from any other function

```
int CalculateSum(int a, int b);

int main()
{
  int r{};
  r = CalculateSum(2, 4);
}

int CalculateSum(int a, int b)
{
  return a + b;
}
```

Contents

- How to use functions
- Function value parameters
- Return by value statement
- default parameters
- Function reference parameters
- return by reference
- Function overloading and ambiguity
- Call stack

Arguments and parameters

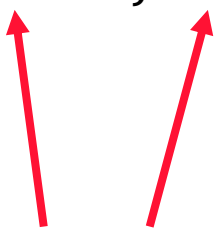
```
int CalculateSum(int a, int b);

int main()
{
  int r{};
  r = CalculateSum(2, 4);
}

int CalculateSum(int a, int b)
{
  return a + b;
}
```

A parameter is a value that is passed from the caller function to the function. Different parameters are separated by commas.

An argument is a local variable of the function where the value is provided by the caller. Different arguments are separated by commas.

Arguments and parameters

```
int CalculateSum(int a, int b);

int main()
{
  int r{};
  r = CalculateSum(2, 4);
}

int CalculateSum(int a, int b)
{
  return a + b;
}
```



The type of the parameters used when calling the function must match the type in the prototype and definition.

Pass by value

The value of each parameter is copied into the matching argument. (order)

```
int CalculateSum(int a, int b);

int main()
{
  int r{};
  r = CalculateSum(2, 4);
}

int CalculateSum(int a, int b)
{
  return a + b;
}
```

Arguments are local variables of the function, they are created when the function starts, and destroyed when the function ends.

return value

The evaluation of the optional return expression is copied to the callers lvalue (if there is one)

```
int CalculateSum(int a, int b);

int main()
{
  int r{};
  r = 6;
}

int CalculateSum(int a, int b)
{
  return 2 + 4;
}
```

When the return type is not void, a return statement is obligatory
The returned value/variable must match the return type

return value is optional: void

No assignment

```
void PrintValues(int a, int b);

int main()
{
  PrintValue(4, 8);
}

void PrintValue(int a, int b)
{
  std::cout << a << "   " << b;
}
```

No mandatory return statement when the return type is void

return statement:

The return statement causes the immediate exit of the function.
Be careful, nothing is printed in this example

```
void PrintValues(int a, int b);

int main()
{
  PrintValue(4, 8);
}

void PrintValue(int a, int b)
{
  return;
  std::cout << a << " " << b;
}
```

return statement:

Using return to terminate the execution of a function:

```
void Count(int number)
{
  int count = 0;
  while (count < 50)
  {
    ++count;
    if (rand() == number)
    {
      std::cout << "Number found after " << count << " iterations.";
      return;
    }
  }
  std::cout << "Number not found in 50 iterations";
}
```

Return value: existing examples

```
double result = std::sqrt(44);
double result = std::pow(2,4);
```

What is unexpected?

Return value: existing examples

```
double result = std::sqrt(44);
double result = std::pow(2,4);
```

The int parameters are typecasted to double. See:
http://en.cppreference.com/w/cpp/numeric/math/pow

Guidelines

- Choose an identifier that describes the functionality of the function.
- Start the identifier with a verb.
  - Example: int CalculateSum(int a, int b)
    - This function does NOT print the result

Guidelines

- Carefully choose the type and name of the different parameters.
- Consider wisely whether a return value is desired, and what type is most appropriate.

Guidelines

- Code that appears more than once in a program should be made in a function.
- A function should generally perform only one task.
- When a function becomes too long, too complicated or hard to understand, it should be split into multiple sub-functions (refactoring).

Why?

- Organization
- Reusability
- Testing
- Extensibility
- Abstraction

Contents

- How to use functions
- Function value parameters
- Return by value statement
- Function overloading and ambiguity
- Call stack

Related concepts

1) The concept of overloading
2) The concept of ambiguity

Overloading: what?

- "overloading" of a function means: to have more than one function with the same name.
- Problem: how does the C++ compiler know which function to choose?
- Example:

```
int CalculateSum( int number1);
int CalculateSum( int number1, int number2);
```

Overloading: what function version?

- When a function is called, C++ automatically deduces the function using the following aspects:
  - name (identifier) of the function
  - number of parameters
  - type of the parameters
  - order of the parameters
- NOT the return type, NOT the name of parameters

Ambiguity

- What is it?
- Origin: Latin: “open to different interpretations”.
- 2 overloaded functions can be ambiguous.
- The compiler can not choose between two overloaded functions.
- Ambiguous functions result in compile errors.

Ambiguity and default parameters

- Be careful: using default parameters can result in ambiguity: compile error

```
double SomeFunction (double number1, int number2 = 100);
double SomeFunction (double number1);
```

```
: ambiguous call to overloaded function
```

Exercises

- Are the functions of which the declarations are given here, overloaded versions of the same function, or ambiguous?

!! Ambiguous !!

```
void PrintDate( int monthNumber );

void PrintDate( int yearNumber );
```

!! Ambiguous !!

```
int PrintDate( int monthNumber );

void PrintDate( int yearNumber );
```

Overloading Ok

```
int Subtraction( int number1, int number2 );

double Subtraction( double number1, double number2 );
```

!! Ambiguous !!

```
int Product(int number1, int number2 );

double Product(int number1, int number2 );
```

Overloading Ok

```
int Product(float number1, int number2 );

double Product(int number1, float number2 );
```

Contents

- How to use functions
- Function value parameters
- Return by value statement
- Function overloading and ambiguity
- Call stack

Memory organisation

- The memory a program uses:
  - Code area → compiled code area
  - Global area → global and static variables
  - Heap → dynamically allocated memory (see later)
  - Stack → function arguments and local variables

The (call) stack

- What is it? A limited stack of memory defined at compile time.
- When is it accessed? Declaration of local variables and parameters of a function.
- Automatic storage. Storage is claimed automatically at the end of the scope.

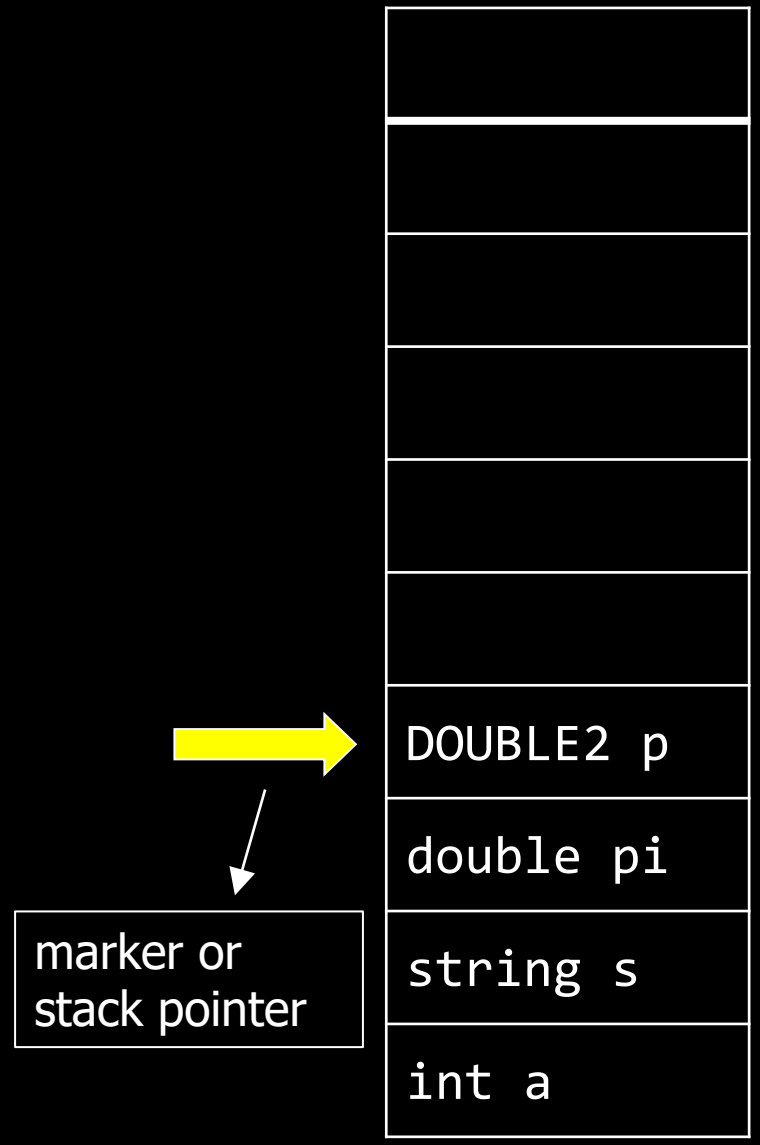
The (call) stack

- Location? → Fixed memory stack (eg. 1 MByte)
- Lifespan? → "stack frame" → depends scope of a variable:
  - A local variable is added to the stack when it is initialized.
  - Automatically removed from the stack when it goes out of scope.

DIGITAL ARTS & ENTERTAINMENT

The (call) stack

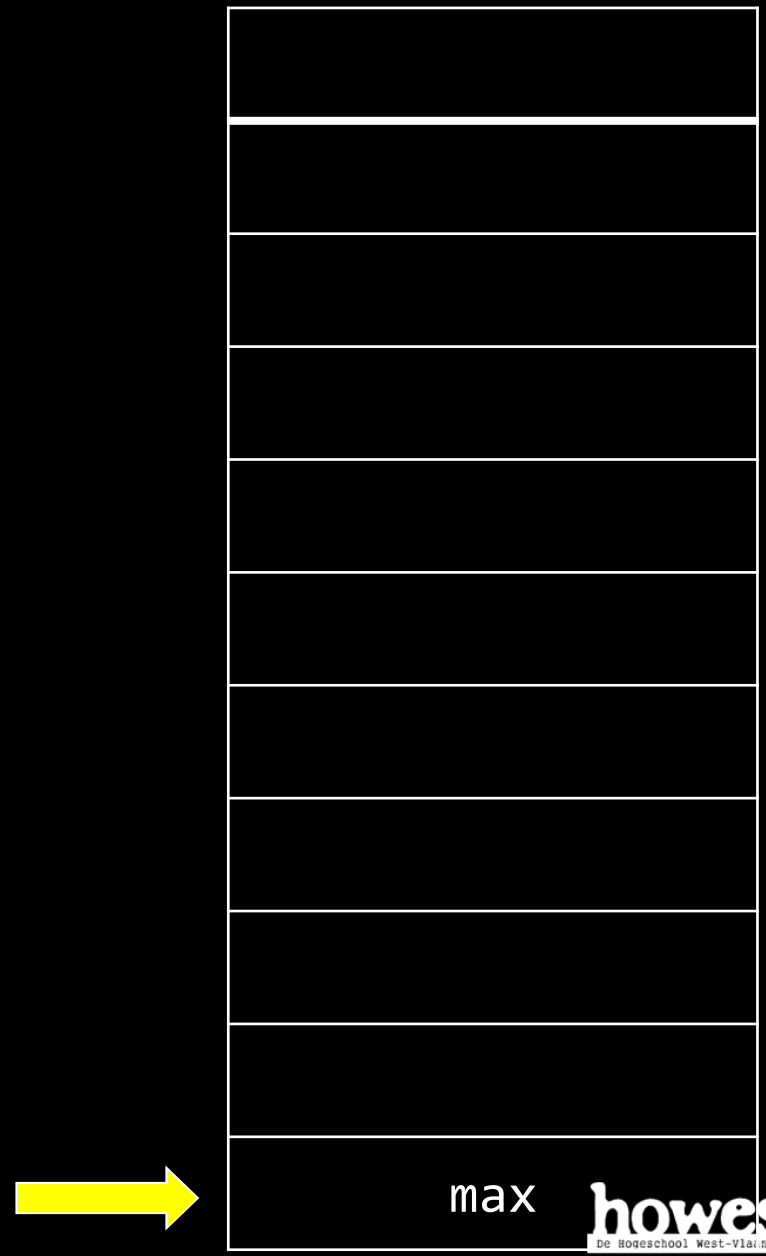
- Mechanics:
  - a container that holds variables
  - LIFO: last in, first out
  - size is fixed
  - 3 possible operations:
    - Look at top → top()
    - Take top item off the stack → pop()
    - Put a new item on the stack → push()
  - Stack pointer keeps track of top



howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

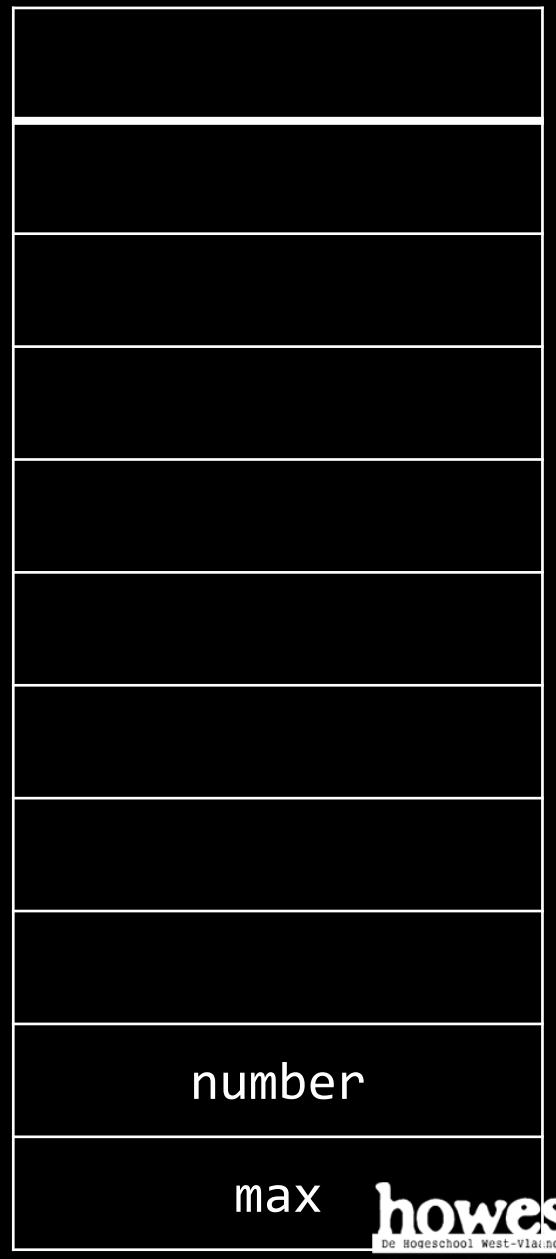
The stack in Action

```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```



The stack in Action

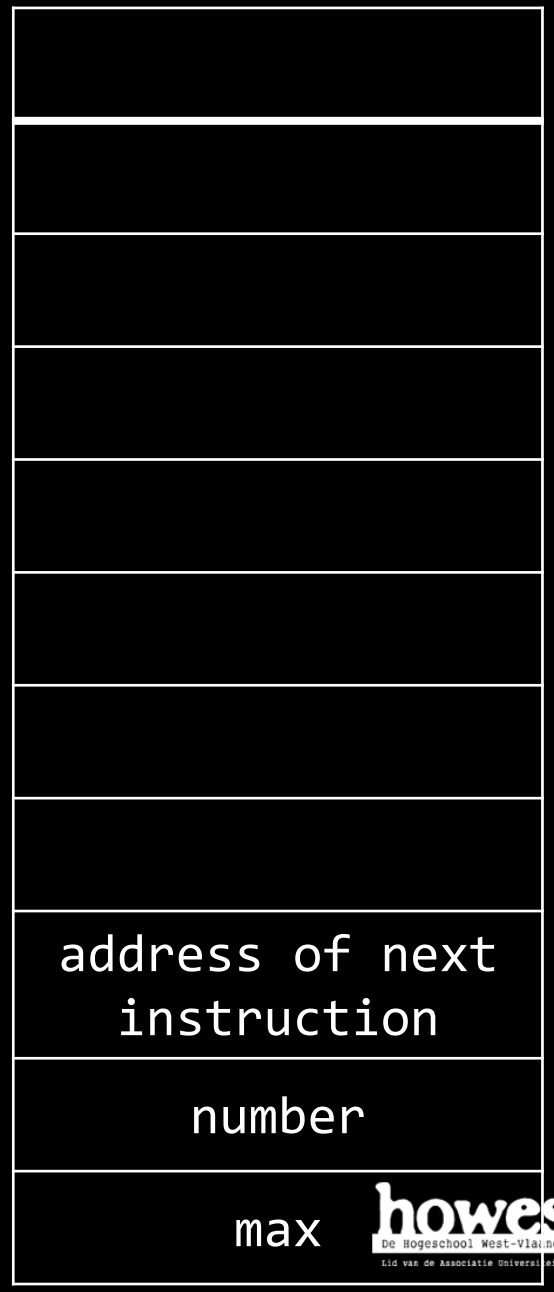
```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```



DIGITAL ARTS & ENTERTAINMENT

The stack in Action

```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```

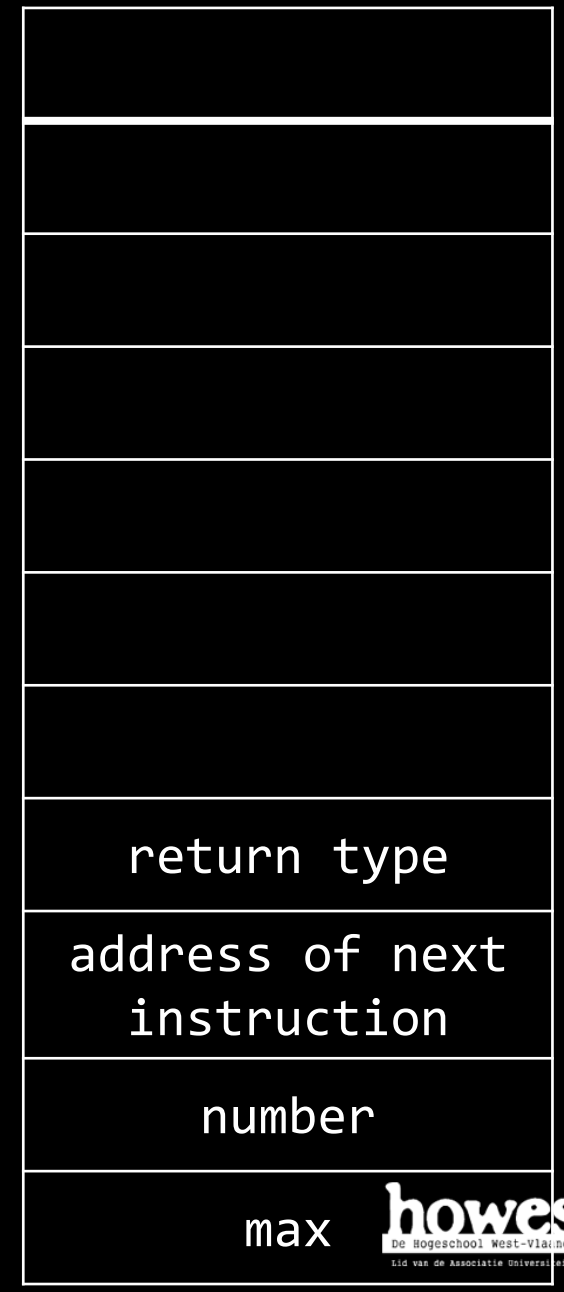


howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

DIGITAL ARTS & ENTERTAINMENT

The stack in Action

```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```



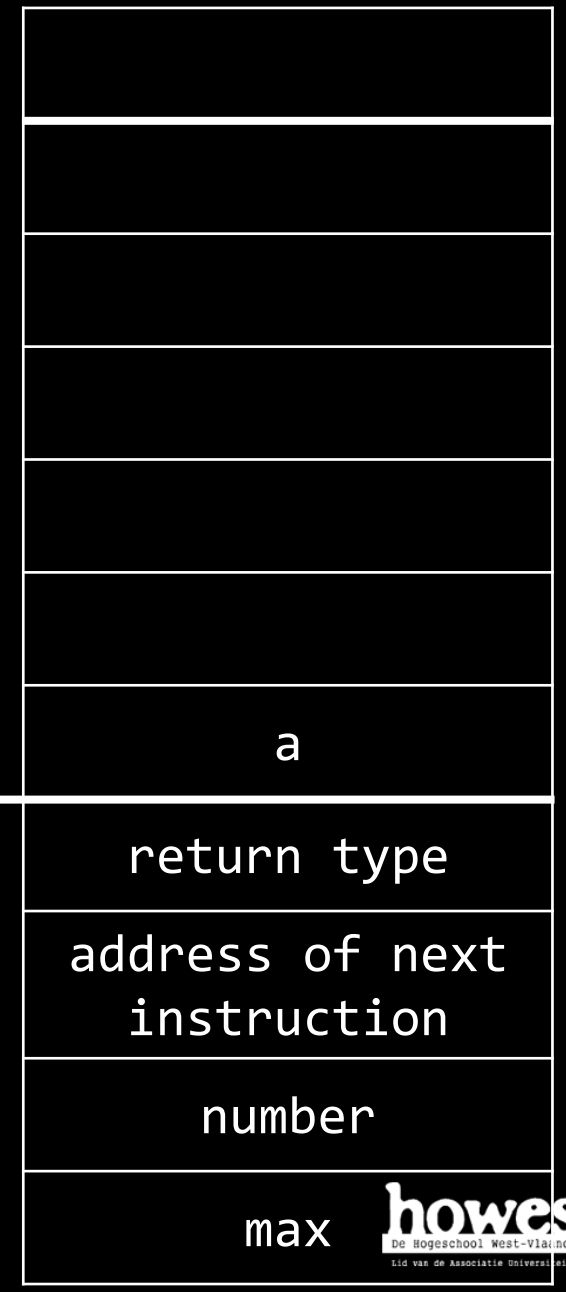
howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

DIGITAL ARTS & ENTERTAINMENT

The stack in Action

```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```

stack frame

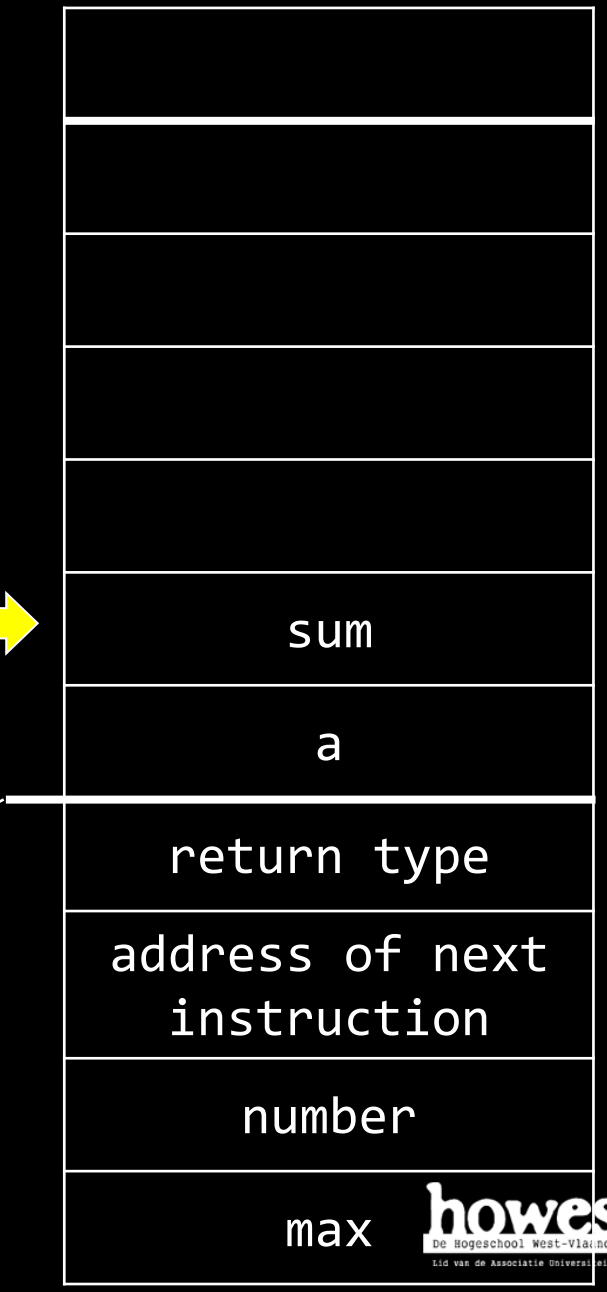


howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

DIGITAL ARTS & ENTERTAINMENT

The stack in Action

```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```

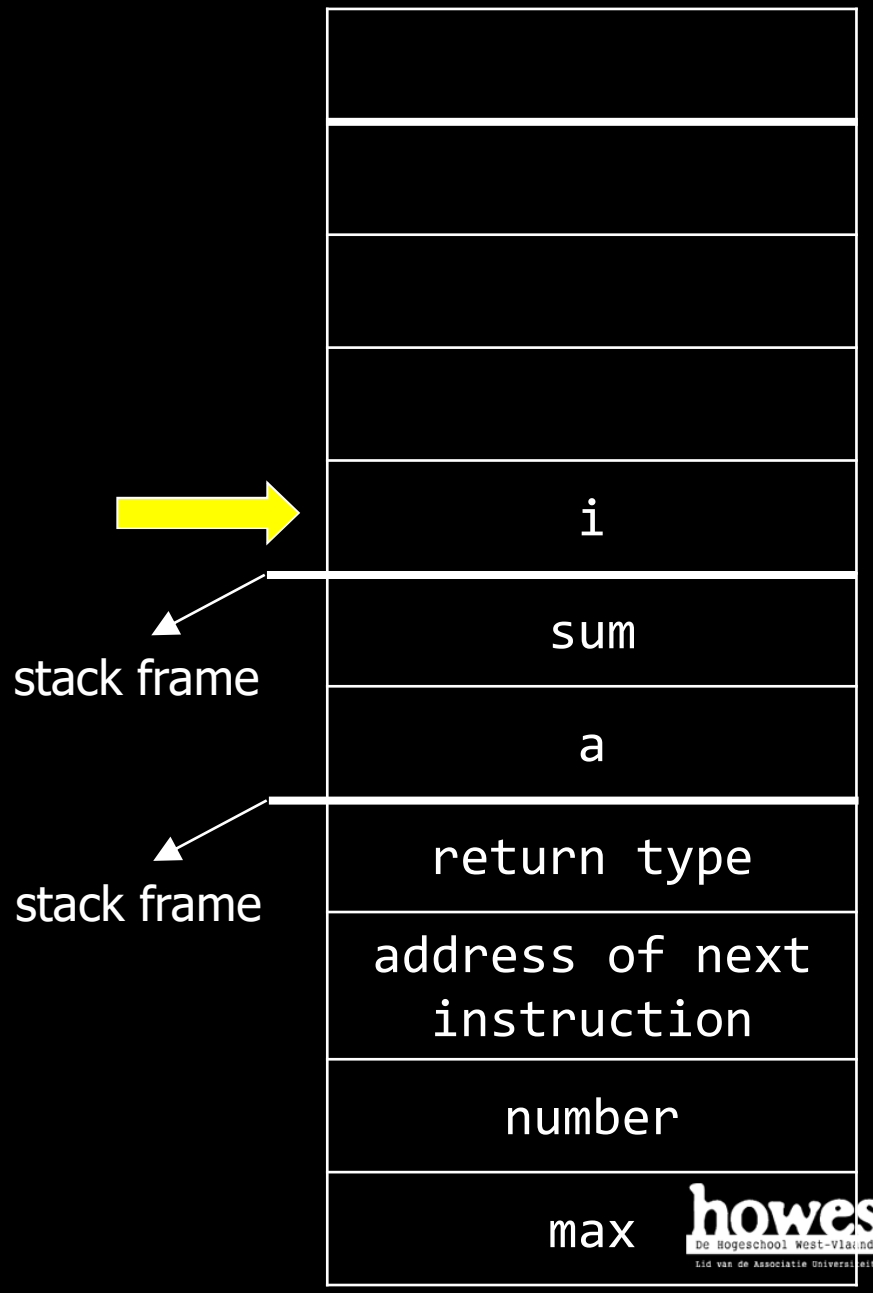


stack frame

howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

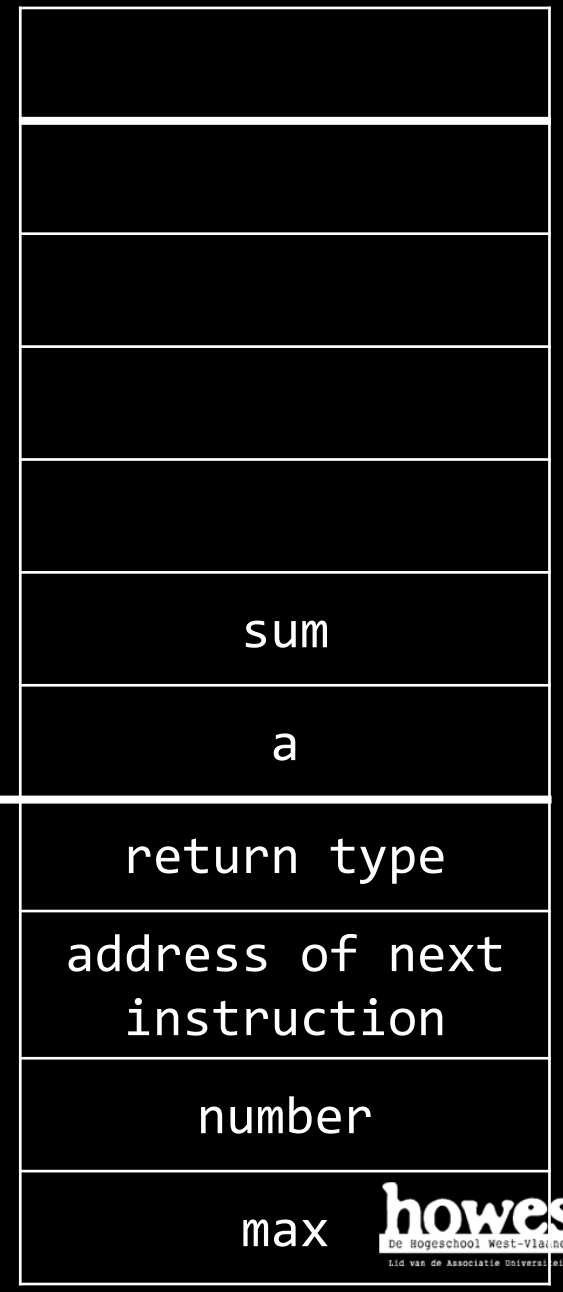
The stack in Action

```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```



The stack in Action

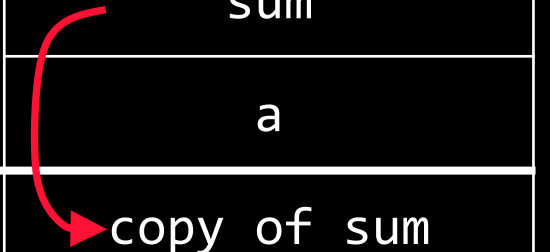
```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```



DIGITAL ARTS & ENTERTAINMENT

The stack in Action

```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```



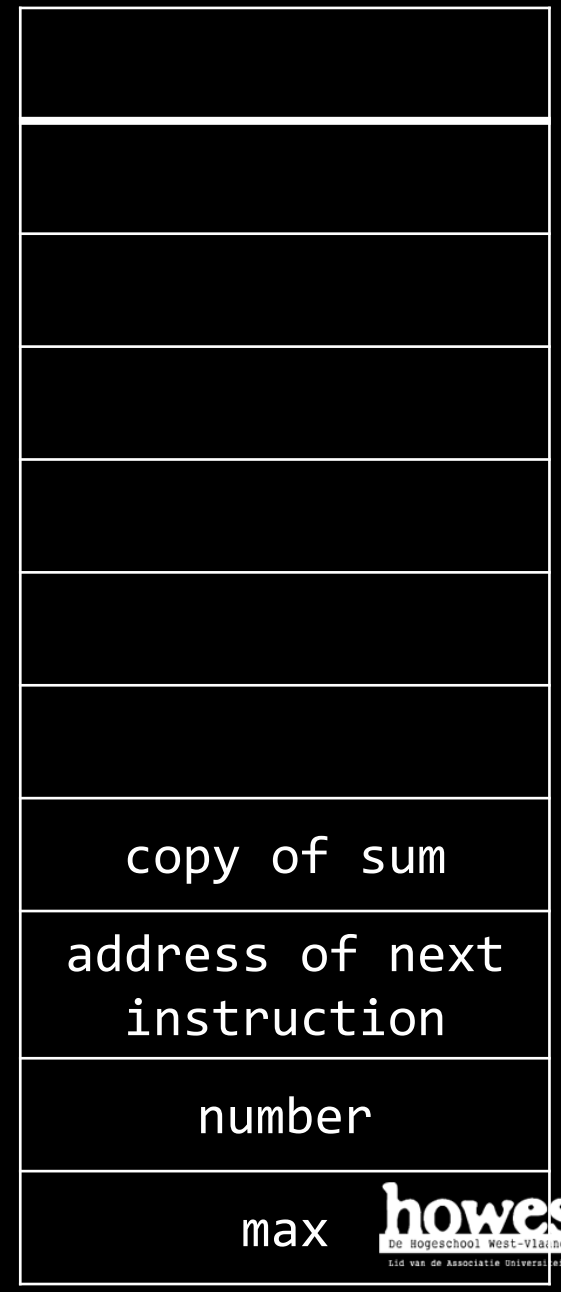
stack frame

howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

DIGITAL ARTS & ENTERTAINMENT

The stack in Action

```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```

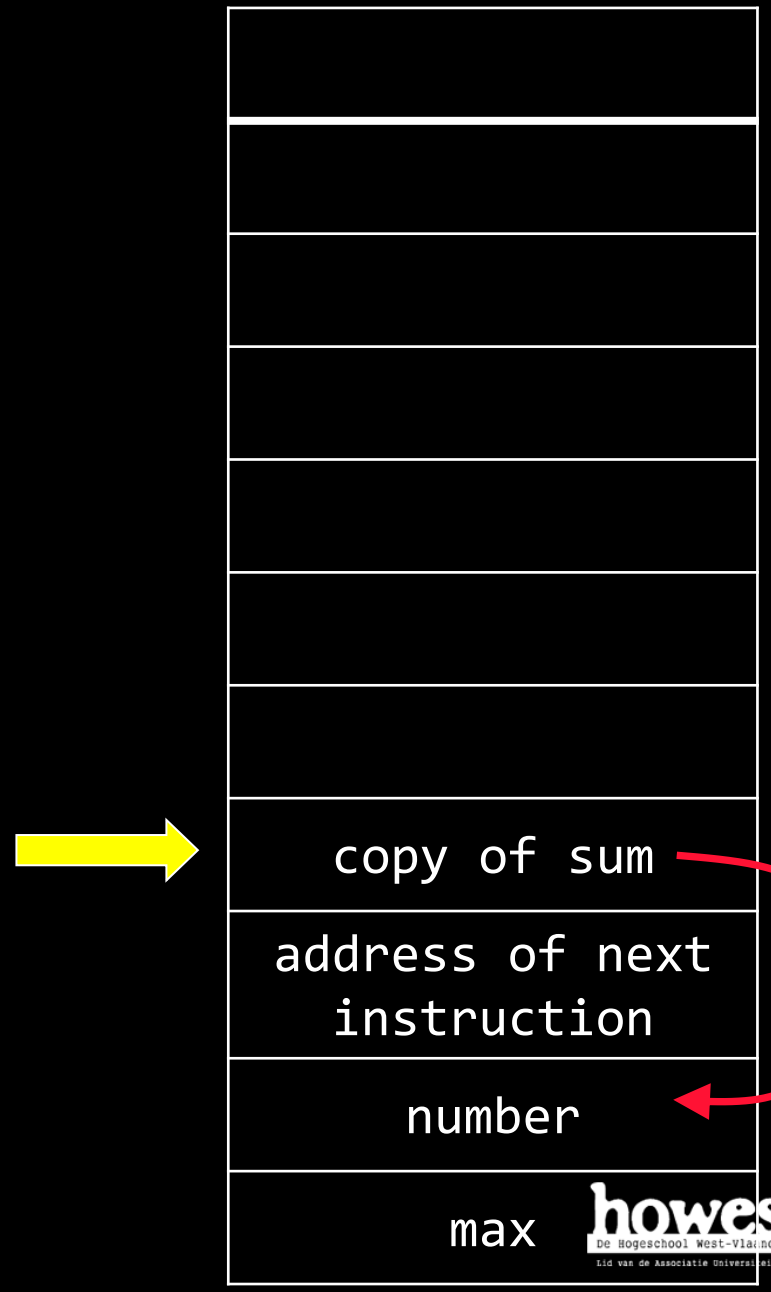


howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

DIGITAL ARTS & ENTERTAINMENT

The stack in Action

```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```

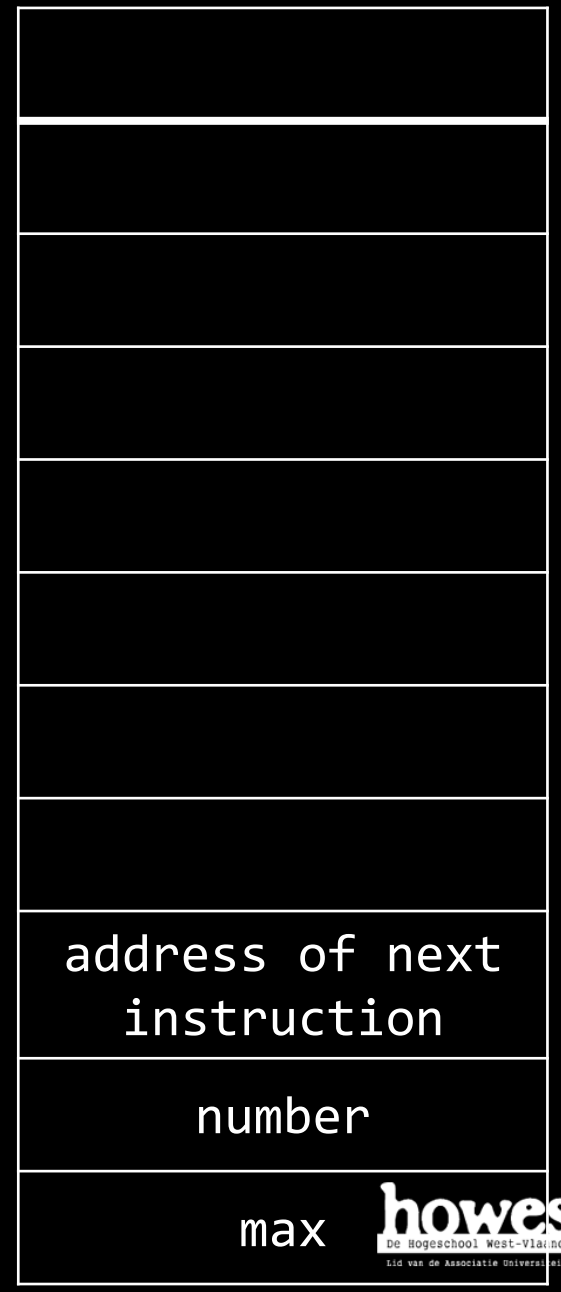


howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

DIGITAL ARTS & ENTERTAINMENT

The stack in Action

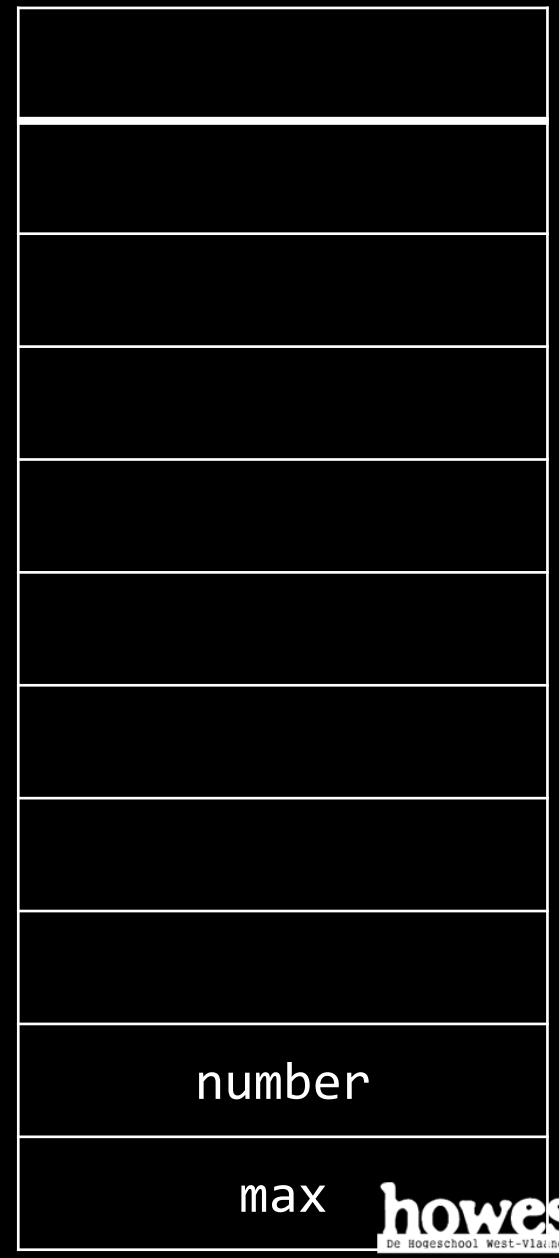
```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```



howest
De Hogeschool West-Vlaanderen
Lid van de Associatie Universiteit Gent

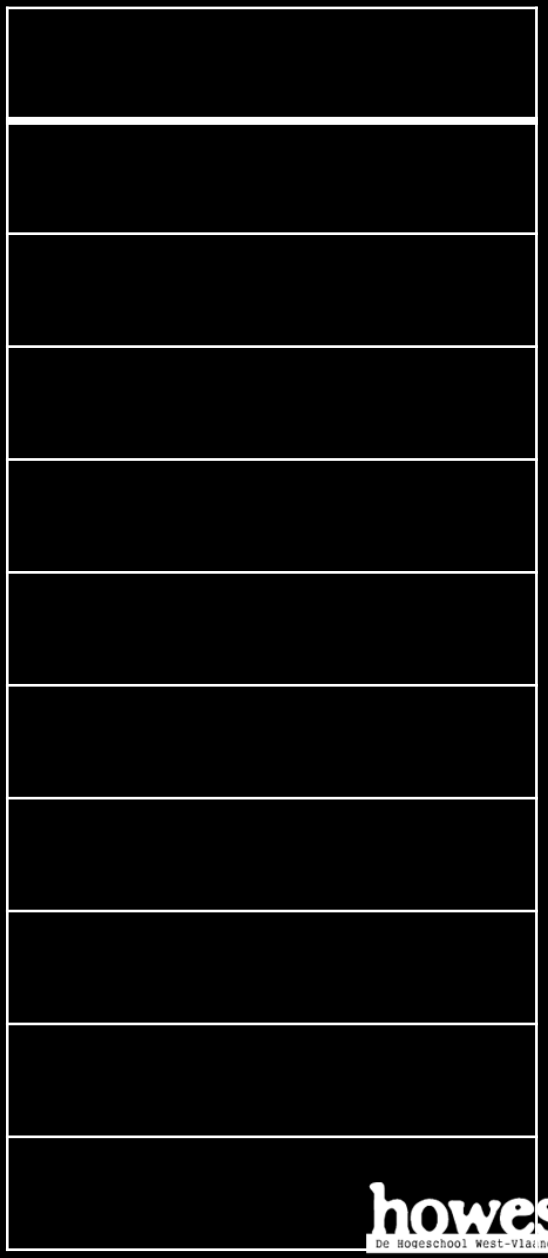
The stack in Action

```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```



The stack in Action

```
int main(int argc, char *argv[])
{
  int max = 10;
  int number = Sum(max);
  cout << number;
}

int Sum(int a)
{
  int sum = 0;
  for (int i = 0; i < a; i++)
  {
    sum += i;
  }
  return sum;
}
```



References

- http://www.learncpp.com/cpp-tutorial/71-function-parameters-and-arguments/