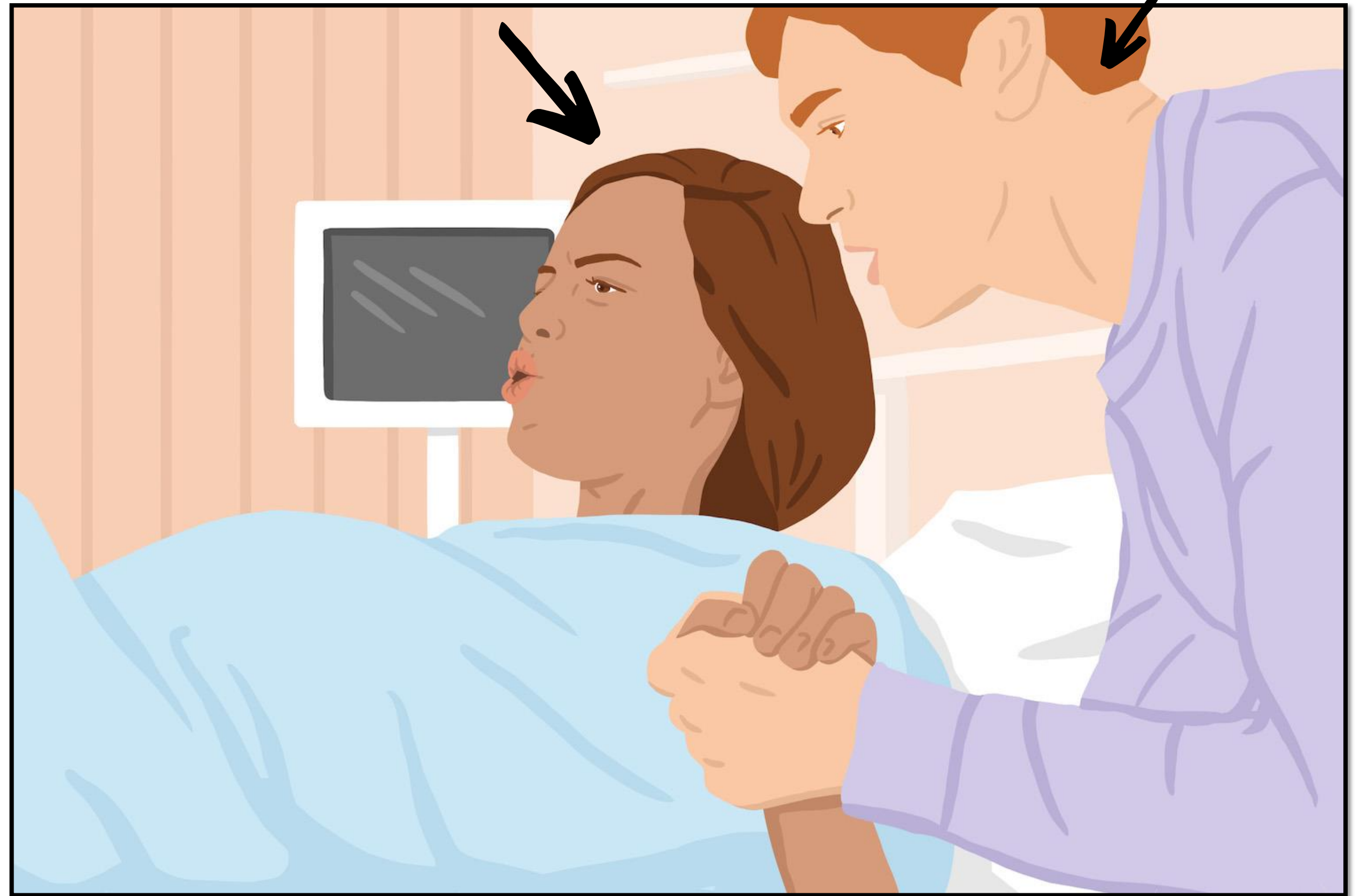


In 1958 and
1960...

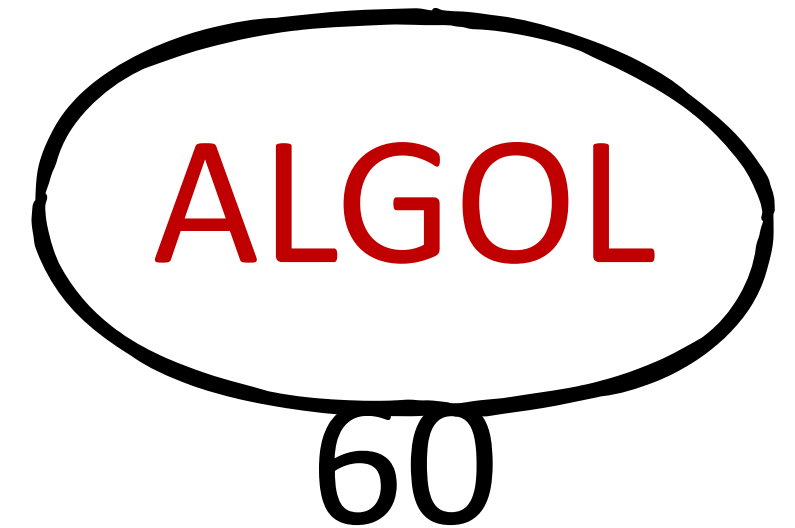
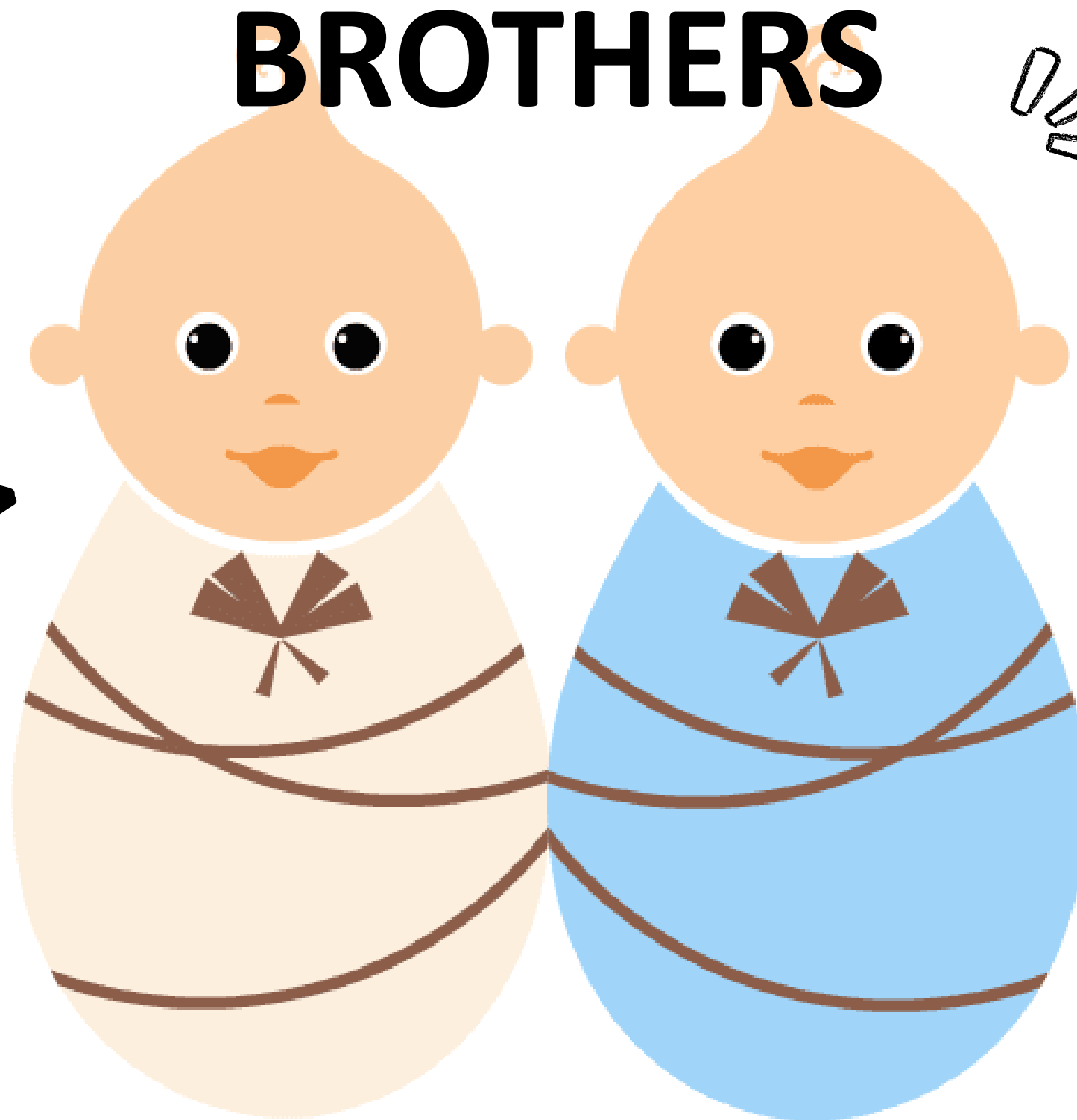
European CS

American CS



ALGORITHMIC LANGUAGE

BROTHERS





**WHAT MAKES THESE
ALGOL BROTHERS**

SPECIAL?

ALGOL'S PROGRAMMING LANGUAGE

INNOVATIONS

Nested
Block

Structure

↑
Powerful tool in building
larger programs out of
smaller components

Lexical
Scoping

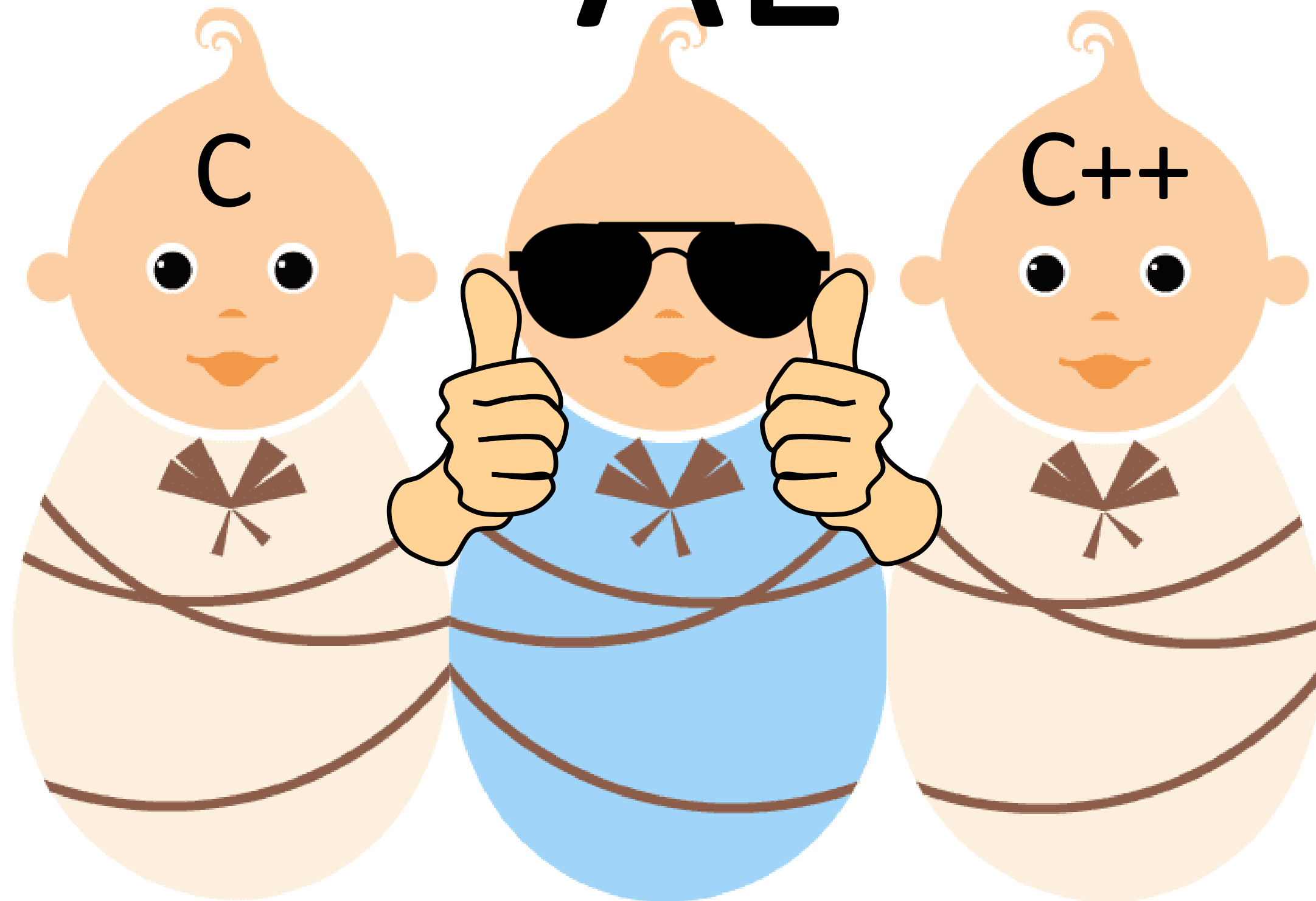
↑
Useful in information
hiding

Backus–
Naur Form

(BNF)

↑
Became standard tool for
stating the grammar of
programming languages

PASC AL



TEAM OOP PRESENTS

PASCAL Program ming

for dummies



HISTORY

- is an imperative and procedural programming language
- designed by Niklaus Wirth
- grew out of ALGOL 60
- Wirth was part of ALGOL X and he proposed ALGOL W where there are strings and has cleaned syntax
- released ALGOL W as Pascal in 1970

Who's laughing
now, huh?



Niklaus Wirth

HISTORY


- Named after French mathematician and philosopher **Blaise Pascal**
- useful for teaching **structured programming** and **data structuring**



Blaise Pascal




REASONS WHY PASCAL WAS POPULAR:

- Easy to learn.
 - Structured language.
 - Produces transparent, efficient and reliable programs.
 - Can be compiled on a variety of computer platforms.
- 



PASCAL'S SUCCESS:

- Pascal became widely accepted at universities in early 1980s
 - Introduction of **UCSD Pascal** in **Apple II** which led Pascal in becoming primary high-level language used in **Apple Lisa**
 - Introduction of **Object Pascal** on the Mac which became Apple's main development language into the early 1990s
 - Introduction of **Turbo Pascal**
- 


Domain:

Education, early
microcontrollers

Paradigm:

Imperative-structured





```
program exFunction;
var
  a, b, ret : integer;

(*function definition *)
function max(num1, num2: integer): integer;
var
  (* local variable declaration *)
  result: integer;

begin
  if (num1 > num2) then
    result := num1

  else
    result := num2;
  max := result;
end;

begin
  a := 100;
  b := 200;
  (* calling a function to get max value *)
  ret := max(a, b);

  writeln( 'Max value is : ', ret );
end.
```

Paradigms:

- Structured
- Imperative
- Procedural

OPERATORS

- Arithmetic operators
- Relational operators
- Boolean operators
- Bit operators
- Set operators
- String operators



ARITHMETIC OPERATORS



Operator	Description	Example
+	Adds two operands	$A + B$ will give 30
-	Subtracts second operand from the first	$A - B$ will give -10
*	Multiplies both operands	$A * B$ will give 200
/	Divides numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	$B \% A$ will give 0

RELATIONAL OPERATORS



Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes, then condition becomes true.	(A = B) is not true.
<>	Checks if the values of two operands are equal or not, if values are not equal, then condition becomes true.	(A <> B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes, then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes, then condition becomes true.	(A <= B) is true.

BOOLEAN OPERATORS



Operator	Description	Example
and	Called Boolean AND operator. If both the operands are true, then condition becomes true.	(A and B) is false.
and then	It is similar to the AND operator, however, it guarantees the order in which the compiler evaluates the logical expression. Left to right and the right operands are evaluated only when necessary.	(A and then B) is false.
or	Called Boolean OR Operator. If any of the two operands is true, then condition becomes true.	(A or B) is true.
or else	It is similar to Boolean OR, however, it guarantees the order in which the compiler evaluates the logical expression. Left to right and the right operands are evaluated only when necessary.	(A or else B) is true.
not	Called Boolean NOT Operator. Used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	not (A and B) is true.

BIT OPERATORS



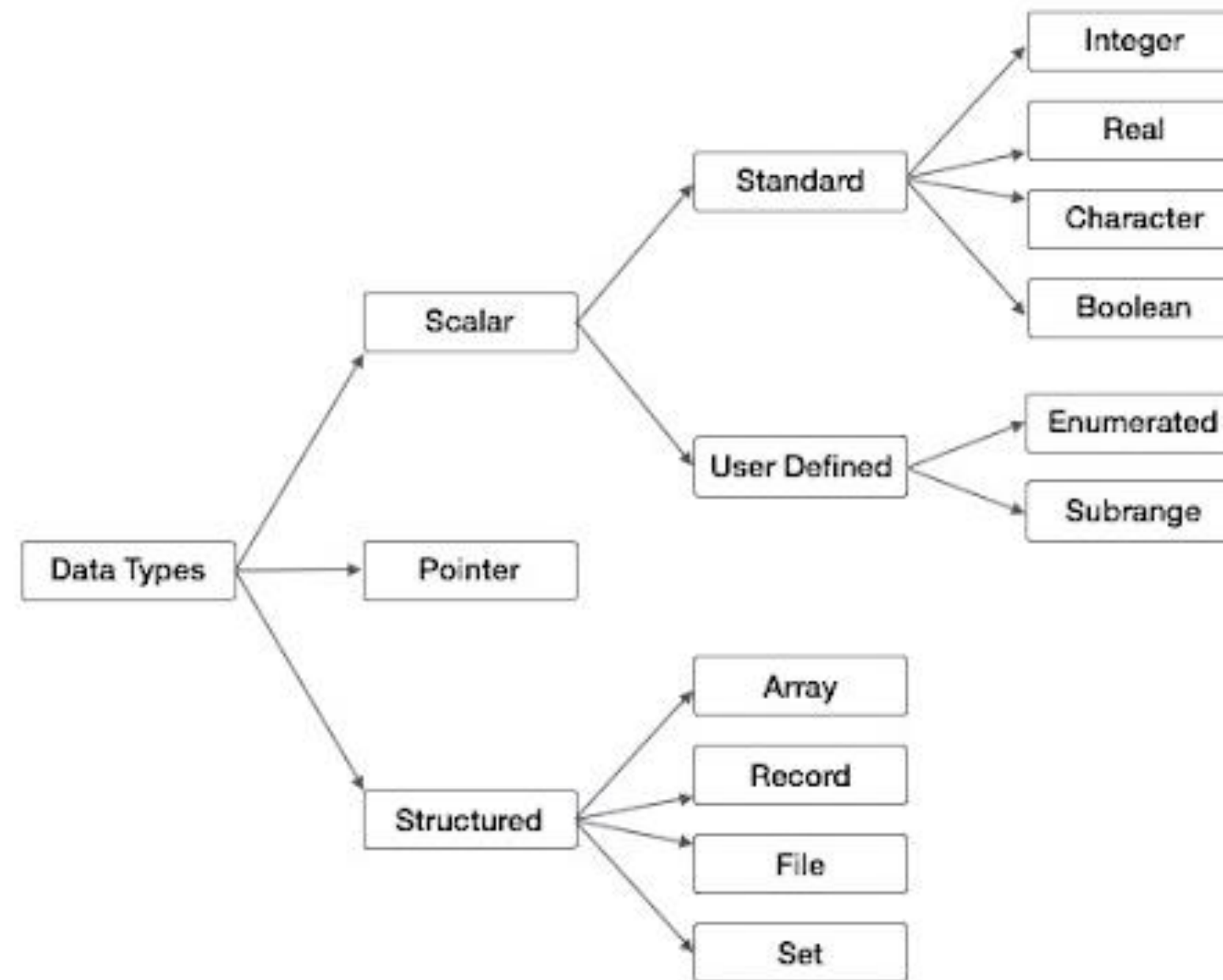
Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12, which is 0000 1100
	Binary OR Operator copies a bit if it exists in either operand.	(A B) will give 61, which is 0011 1101
!	Binary OR Operator copies a bit if it exists in either operand. Its same as operator.	(A ! B) will give 61, which is 0011 1101
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61, which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 0000 1111

BIT OPERATORS






Operators	Operations
not	Bitwise NOT
and	Bitwise AND
or	Bitwise OR
xor	Bitwise exclusive OR
shl	Bitwise shift left
shr	Bitwise shift right
<<	Bitwise shift left
>>	Bitwise shift right

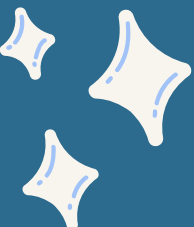
DATA STRUCTURE





SIGNED DATA TYPES

- Int8 (-128 .. 127)
 - ShortInt (-128 .. 127)
 - Int16 (-32768 .. 32767)
 - SmallInt (-32768 .. 32767)
 - Integer - equivalent either to Smallint or Longint
 - Int32 (-2147483648 .. 2147483647)
 - LongInt (-2147483648 .. 2147483647)
 - Int64 (-9223372036854775808 to 9223372036854775807)
- 
- 
- 



UNSIGNED DATA TYPES

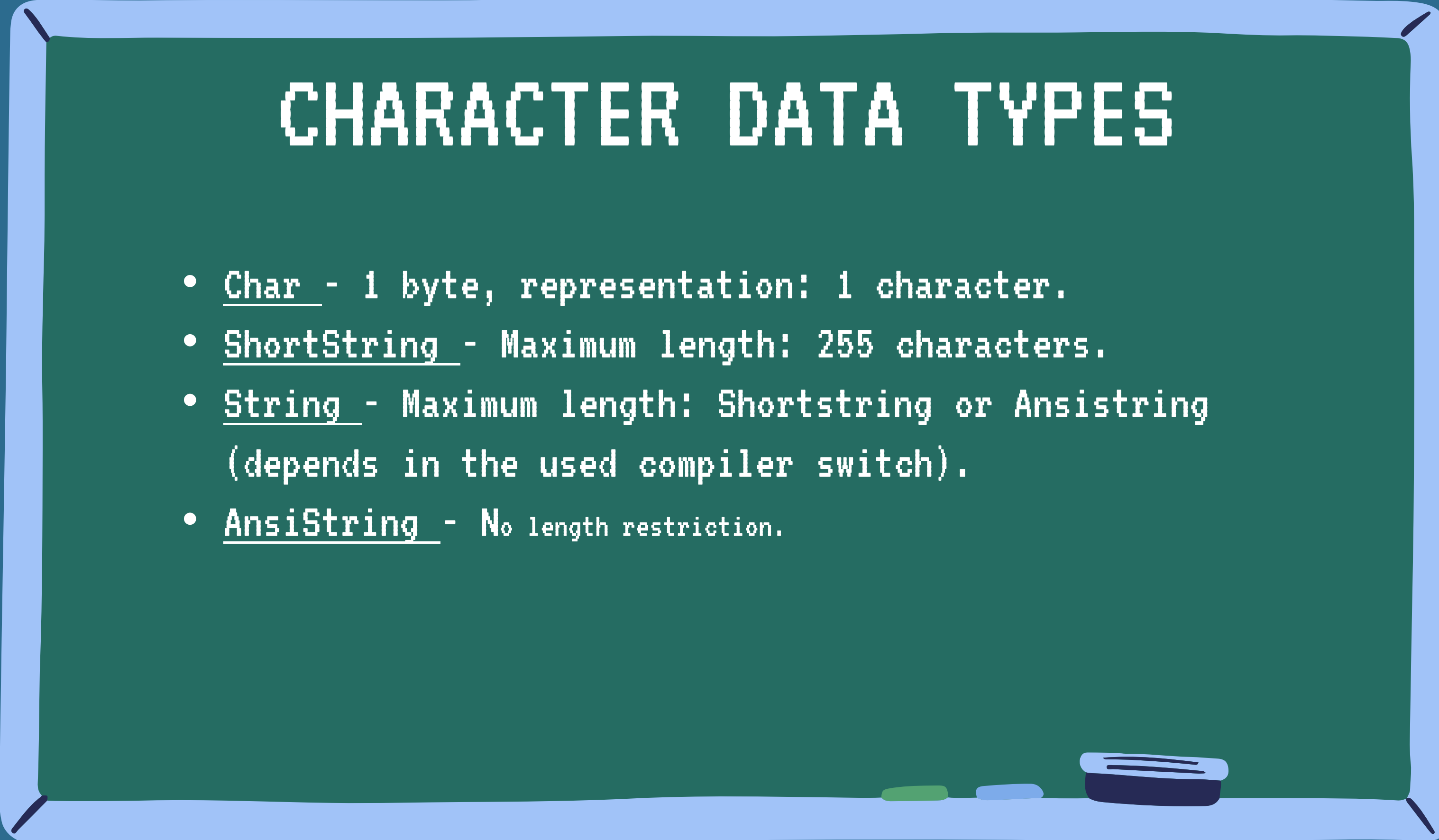
- UInt8 (0 .. 255)
- Byte (0 .. 255)
- UInt16 (0 .. 65535)
- Word - (0 .. 65535)
- UInt32 (0 .. 4294967295)
- Longword (0 .. 4294967295)
- UInt64 (0 .. 18446744073709551615)
- QWord (0 .. 18446744073709551615)

*only positive values allowed



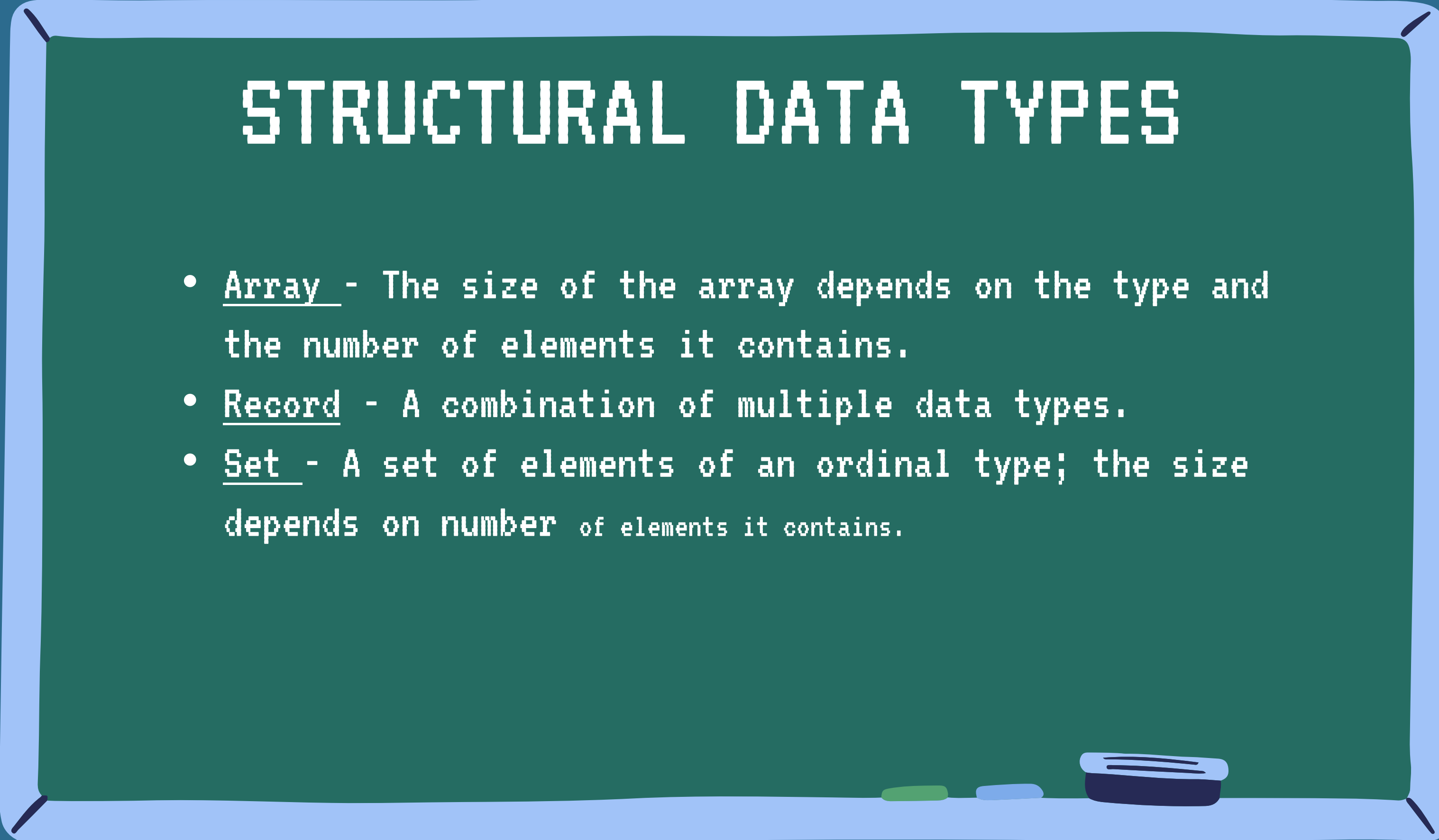


CHARACTER DATA TYPES

- Char - 1 byte, representation: 1 character.
 - ShortString - Maximum length: 255 characters.
 - String - Maximum length: Shortstring or Ansistring (depends in the used compiler switch).
 - AnsiString - No length restriction.
- 

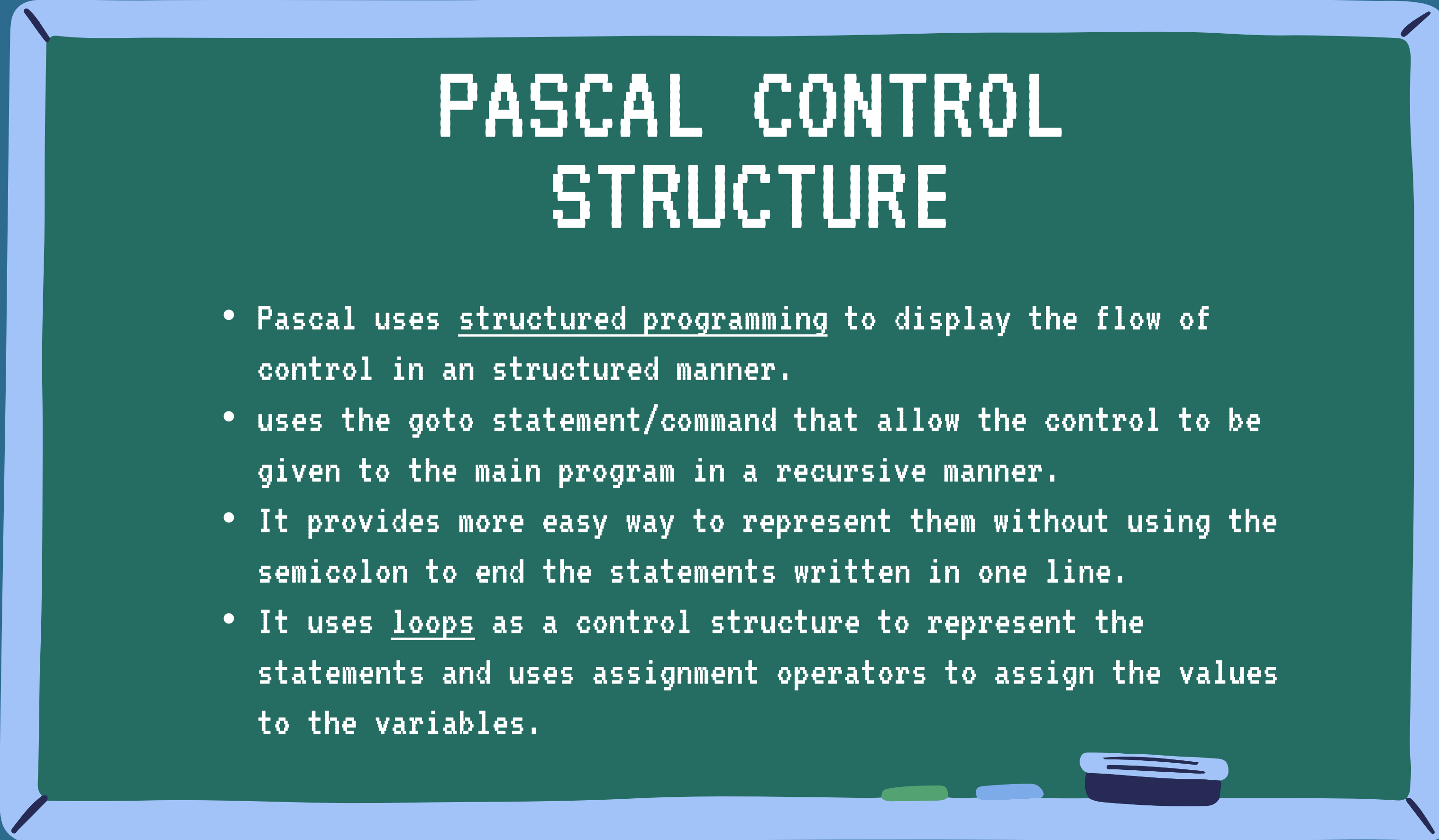


STRUCTURAL DATA TYPES

- Array - The size of the array depends on the type and the number of elements it contains.
 - Record - A combination of multiple data types.
 - Set - A set of elements of an ordinal type; the size depends on number of elements it contains.
- 



PASCAL CONTROL STRUCTURE

- Pascal uses structured programming to display the flow of control in an structured manner.
 - uses the goto statement/command that allow the control to be given to the main program in a recursive manner.
 - It provides more easy way to represent them without using the semicolon to end the statements written in one line.
 - It uses loops as a control structure to represent the statements and uses assignment operators to assign the values to the variables.
- 

C Code

```
#include <stdio.h>
int factorialIterative(int n){
    int product = 1;
    while(n > 0){
        product *= n--;
    }
    return product;
}
```

```
int factorialRecursive(int n){
    if(n == 1) return 1;
    else
        return
        factorialRecursive(n-1) *
        n;
}
int main(){
    int n; scanf("%d", &n);
    int out =
    factorialRecursive(n);
```

```
    printf("%d", out);
```

PASCAL Code

```
program Hello;
```

```
var
```

```
    n : integer;
```

```
function factorialIterative(n :  
integer) : integer;
```

```
var
```

```
    product : integer;
```

```
begin
```

```
    product := 1;
```

```
    while n > 0 do
```

```
        begin
```

```
            product := product * n;
```

```
            n := n - 1;
```

```
        end;
```

```
        factorialIterative :=
```

```
product;
```

```
end;
```


PASCAL Code

```
function factorialRecursive(nbegin
: integer) : integer;           n := 7;
begin                           writeln
    if n = 1 then               (factorialIterative(n))
        factorialRecursive := 1 end.
    else
        factorialRecursive :=
factorialRecursive(n-1) * n;
end;
```