AP Computer Science Principles Pseudocode

For C++ Students

Mr. Gullo

November 2025

Learning Objectives

After this lesson, you will be able to:

- Explain what pseudocode is and why it is used
- Read and understand AP CSP pseudocode format
- Convert simple C++ code to AP CSP pseudocode
- Convert AP CSP pseudocode to C++ code
- Identify key differences between pseudocode and C++ syntax

What is Pseudocode?

Pseudocode is a way to write algorithms using simple, plain language.

What is Pseudocode?

Pseudocode is a way to write algorithms using simple, plain language.**Key Points:**

- Not a real programming language
- Cannot run on a computer
- Easy to read and understand
- Focuses on logic, not syntax
- Used on AP CSP exam

Why Learn AP CSP Pseudocode?

Reasons:

- Required for AP Computer Science Principles exam
- Language-independent (works for any programming language)
- Easier to communicate ideas
- Less strict than real code
- Helps focus on problem-solving

Why Learn AP CSP Pseudocode?

Reasons:

- Required for AP Computer Science Principles exam
- Language-independent (works for any programming language)
- Easier to communicate ideas
- Less strict than real code
- Helps focus on problem-solving

Good News: If you know C++, learning pseudocode is easy!

Variables and Assignment

AP CSP Pseudocode:

```
x <- 5
name <- "Alice"
isStudent <- true
```

Variables and Assignment

AP CSP Pseudocode:

```
x <- 5
name <- "Alice"
isStudent <- true
```

C++ Equivalent:

```cpp
int x = 5;
string name = "Alice";
bool isStudent = true;
```

Variables and Assignment

AP CSP Pseudocode:

```
x <- 5
name <- "Alice"
isStudent <- true
```

C++ Equivalent:

```cpp
int x = 5;
string name = "Alice";
bool isStudent = true;
```

Key Difference: Use $\leftarrow$ instead of $=$

Display Output

AP CSP Pseudocode:

```
DISPLAY("Hello World")
DISPLAY(x)
DISPLAY("Your score is: ", score)
```

Display Output

AP CSP Pseudocode:

```
DISPLAY("Hello World")
DISPLAY(x)
DISPLAY("Your score is: ", score)
```

C++ Equivalent:

```
cout << "Hello World" << endl;
cout << x << endl;
cout << "Your score is: " << score << endl;
```

Display Output

AP CSP Pseudocode:

```
DISPLAY("Hello World")
DISPLAY(x)
DISPLAY("Your score is: ", score)
```

C++ Equivalent:

```
cout << "Hello World" << endl;
cout << x << endl;
cout << "Your score is: " << score << endl;
```

Key Difference: Use DISPLAY() instead of cout

Input from User

AP CSP Pseudocode:

```
age <- INPUT()
name <- INPUT()
```

Input from User

AP CSP Pseudocode:

```
age <- INPUT()
name <- INPUT()
```

C++ Equivalent:

```
int age;
cin >> age;
string name;
cin >> name;
```

Input from User

AP CSP Pseudocode:

```
age <- INPUT()
name <- INPUT()
```

C++ Equivalent:

```
int age;
cin >> age;
string name;
cin >> name;
```

Key Difference: Use INPUT() instead of cin

Comments

AP CSP Pseudocode:

```
// This is a comment
x <- 10  // Set x to 10
```

Comments

AP CSP Pseudocode:

```
// This is a comment
x <- 10  // Set x to 10
```

C++ Equivalent:

```
// This is a comment
int x = 10;  // Set x to 10
```

Comments

AP CSP Pseudocode:

```
// This is a comment
x <- 10  // Set x to 10
```

C++ Equivalent:

```
// This is a comment
int x = 10;  // Set x to 10
```

Same in both! Use // for comments

IF Statements

AP CSP Pseudocode:

```
IF (age >= 18)
{
DISPLAY("You can vote")
}
```

IF Statements

AP CSP Pseudocode:

```
IF (age >= 18)
{
DISPLAY("You can vote")
}
```

C++ Equivalent:

```
if (age >= 18)
{
cout << "You can vote" << endl;
}
```

IF Statements

AP CSP Pseudocode:

```
IF (age >= 18)
{
DISPLAY("You can vote")
}
```

C++ Equivalent:

```
if (age >= 18)
{
cout << "You can vote" << endl;
}
```

Key Difference: IF is uppercase, parentheses optional

IF-ELSE Statements

AP CSP Pseudocode:

```
IF (score >= 60)
{
DISPLAY("Pass")
}
ELSE
{
DISPLAY("Fail")
}
```

IF-ELSE Statements

AP CSP Pseudocode:

```
IF (score >= 60)
{
DISPLAY("Pass")
}
ELSE
{
DISPLAY("Fail")
}
```

C++ Equivalent:

```
if (score >= 60)
{
cout << "Pass" << endl;
}
else
{
cout << "Fail" << endl;
}
```

Comparison Operators

Same in Both:

- $=$ (equal to)
- $\neq$ (not equal to)
- $<$ (less than)
- $>$ (greater than)
- $\leq$ (less than or equal to)
- $\geq$ (greater than or equal to)

Comparison Operators

Same in Both:

- $=$ (equal to)
- $\neq$ (not equal to)
- $<$ (less than)
- $>$ (greater than)
- $\leq$ (less than or equal to)
- $\geq$ (greater than or equal to)

Example:

```
IF (x = 5)          // equal
IF (y != 0)         // not equal
IF (age >= 18)      // greater or equal
```

REPEAT UNTIL Loop

AP CSP Pseudocode:

```
count <- 1
REPEAT UNTIL (count > 5)
{
DISPLAY(count)
count <- count + 1
}
```

REPEAT UNTIL Loop

AP CSP Pseudocode:

```
count <- 1
REPEAT UNTIL (count > 5)
{
DISPLAY(count)
count <- count + 1
}
```

C++ Equivalent:

```
int count = 1;
while (count <= 5)
{
cout << count << endl;
count = count + 1;
}
```

REPEAT UNTIL Loop

AP CSP Pseudocode:

```
count <- 1
REPEAT UNTIL (count > 5)
{
DISPLAY(count)
count <- count + 1
}
```

C++ Equivalent:

```
int count = 1;
while (count <= 5)
{
cout << count << endl;
count = count + 1;
}
```

Key Difference: REPEAT UNTIL is like while loop

REPEAT n TIMES Loop

AP CSP Pseudocode:

```
REPEAT 5 TIMES
{
DISPLAY("Hello")
}
```

REPEAT n TIMES Loop

AP CSP Pseudocode:

```
REPEAT 5 TIMES
{
DISPLAY("Hello")
}
```

C++ Equivalent:

```
for (int i = 0; i < 5; i++)
{
cout << "Hello" << endl;
}
```

REPEAT n TIMES Loop

AP CSP Pseudocode:

```
REPEAT 5 TIMES
{
DISPLAY("Hello")
}
```

C++ Equivalent:

```
for (int i = 0; i < 5; i++)
{
cout << "Hello" << endl;
}
```

Key Difference: Simpler syntax, no counter variable needed

FOR EACH Loop

AP CSP Pseudocode:

```
numbers <- [10, 20, 30, 40]
FOR EACH num IN numbers
{
DISPLAY(num)
}
```

FOR EACH Loop

AP CSP Pseudocode:

```
numbers <- [10, 20, 30, 40]
FOR EACH num IN numbers
{
DISPLAY(num)
}
```

C++ Equivalent:

```
int numbers[] = {10, 20, 30, 40};
for (int num : numbers)
{
cout << num << endl;
}
```

FOR EACH Loop

AP CSP Pseudocode:

```
numbers <- [10, 20, 30, 40]
FOR EACH num IN numbers
{
DISPLAY(num)
}
```

C++ Equivalent:

```
int numbers[] = {10, 20, 30, 40};
for (int num : numbers)
{
cout << num << endl;
}
```

Key Difference: Very similar to C++ range-based for loop

Lists (Arrays)

AP CSP Pseudocode:

```
scores <- [85, 90, 78, 92]
names <- ["Alice", "Bob", "Carol"]
firstScore <- scores[1]
```

Lists (Arrays)

AP CSP Pseudocode:

```
scores <- [85, 90, 78, 92]
names <- ["Alice", "Bob", "Carol"]
firstScore <- scores[1]
```

C++ Equivalent:

```
int scores[] = {85, 90, 78, 92};
string names[] = {"Alice", "Bob", "Carol"};
int firstScore = scores[0];
```

Lists (Arrays)

AP CSP Pseudocode:

```
scores <- [85, 90, 78, 92]
names <- ["Alice", "Bob", "Carol"]
firstScore <- scores[1]
```

C++ Equivalent:

```
int scores[] = {85, 90, 78, 92};
string names[] = {"Alice", "Bob", "Carol"};
int firstScore = scores[0];
```

IMPORTANT: AP CSP lists start at index 1, C++ arrays start at 0!

List Index Comparison

AP CSP (starts at 1):

```
list <- [10, 20, 30]
list[1] // 10
list[2] // 20
list[3] // 30
```

C++ (starts at 0):

```
int list[] = {10, 20, 30};
list[0] // 10
list[1] // 20
list[2] // 30
```

List Index Comparison

AP CSP (starts at 1):

```
list <- [10, 20, 30]
list[1] // 10
list[2] // 20
list[3] // 30
```

C++ (starts at 0):

```
int list[] = {10, 20, 30};
list[0] // 10
list[1] // 20
list[2] // 30
```

Remember: This is the biggest difference! Always check which system you are using.

List Operations

AP CSP Pseudocode:

```
list <- [10, 20, 30]
APPEND(list, 40)            // Add to end: [10, 20, 30, 40]
INSERT(list, 2, 15)         // Insert at position 2: [10, 15, 20, 30, 40]
REMOVE(list, 3)             // Remove position 3: [10, 15, 30, 40]
length <- LENGTH(list)      // Get size: 4
```

List Operations

AP CSP Pseudocode:

```
list <- [10, 20, 30]
APPEND(list, 40)            // Add to end: [10, 20, 30, 40]
INSERT(list, 2, 15)         // Insert at position 2: [10, 15, 20, 30, 40]
REMOVE(list, 3)             // Remove position 3: [10, 15, 30, 40]
length <- LENGTH(list)      // Get size: 4
```

C++ Equivalent (using vector):

```
vector<int> list = {10, 20, 30};
list.push_back(40);                 // Add to end
list.insert(list.begin()+1, 15); // Insert at position 1 (0-indexed)
list.erase(list.begin()+2);         // Remove position 2 (0-indexed)
int length = list.size();           // Get size
```

Procedures (Functions)

AP CSP Pseudocode:

```
PROCEDURE greet(name)
{
DISPLAY("Hello, ", name)
}
greet("Alice")
```

Procedures (Functions)

AP CSP Pseudocode:

```
PROCEDURE greet(name)
{
DISPLAY("Hello, ", name)
}
greet("Alice")
```

C++ Equivalent:

```
void greet(string name)
{
cout << "Hello, " << name << endl;
}
greet("Alice");
```

Procedures with Return Values

AP CSP Pseudocode:

```
PROCEDURE add(a, b)
{
RETURN (a + b)
}
sum <- add(5, 3)
DISPLAY(sum)
```

Procedures with Return Values

AP CSP Pseudocode:

```
PROCEDURE add(a, b)
{
RETURN (a + b)
}
sum <- add(5, 3)
DISPLAY(sum)
```

C++ Equivalent:

```
int add(int a, int b)
{
return (a + b);
}
int sum = add(5, 3);
cout << sum << endl;
```

Exercise 1: Variable Assignment

Convert this C++ code to AP CSP pseudocode:

```
int age = 16;
string name = "Maria";
bool isStudent = true;
cout << "Name: " << name << endl;
cout << "Age: " << age << endl;
```

Exercise 1: Variable Assignment

Convert this C++ code to AP CSP pseudocode:

```
int age = 16;
string name = "Maria";
bool isStudent = true;
cout << "Name: " << name << endl;
cout << "Age: " << age << endl;
```

Answer:

```
age <- 16
name <- "Maria"
isStudent <- true
DISPLAY("Name: ", name)
DISPLAY("Age: ", age)
```

Exercise 2: IF-ELSE Statement

Convert this C++ code to AP CSP pseudocode:

```
int temperature = 75;
if (temperature >= 70)
{
cout << "It's warm" << endl;
}
else
{
cout << "It's cold" << endl;
}
```

Exercise 2: IF-ELSE Statement

Convert this C++ code to AP CSP pseudocode:

```
int temperature = 75;
if (temperature >= 70)
{
cout << "It's warm" << endl;
}
else
{
cout << "It's cold" << endl;
}
```

Answer:

```
temperature <- 75
IF (temperature >= 70)
{
DISPLAY("It's warm")
}
ELSE
{
DISPLAY("It's cold")
```

Exercise 3: Loop Conversion

Convert this C++ code to AP CSP pseudocode:

```
for (int i = 1; i <= 3; i++)
{
cout << "Count: " << i << endl;
}
```

Exercise 3: Loop Conversion

Convert this C++ code to AP CSP pseudocode:

```
for (int i = 1; i <= 3; i++)
{
cout << "Count: " << i << endl;
}
```

Answer (Option 1):

```
REPEAT 3 TIMES
{
DISPLAY("Count: ", i)
}
```

Exercise 3: Loop Conversion

Convert this C++ code to AP CSP pseudocode:

```
for (int i = 1; i <= 3; i++)
{
cout << "Count: " << i << endl;
}
```

Answer (Option 1):

```
REPEAT 3 TIMES
{
DISPLAY("Count: ", i)
}
```

Answer (Option 2):

```
i <- 1
REPEAT UNTIL (i > 3)
{
DISPLAY("Count: ", i)
i <- i + 1
}
```

Exercise 4: List Operations

Convert this C++ code to AP CSP pseudocode:

```
vector<int> scores = {85, 90, 78};
scores.push_back(92);
for (int score : scores)
{
cout << score << endl;
}
```

Exercise 4: List Operations

Convert this C++ code to AP CSP pseudocode:

```
vector<int> scores = {85, 90, 78};
scores.push_back(92);
for (int score : scores)
{
cout << score << endl;
}
```

Answer:

```
scores <- [85, 90, 78]
APPEND(scores, 92)
FOR EACH score IN scores
{
DISPLAY(score)
}
```

Exercise 4: List Operations

Convert this C++ code to AP CSP pseudocode:

```
vector<int> scores = {85, 90, 78};
scores.push_back(92);
for (int score : scores)
{
cout << score << endl;
}
```

Answer:

```
scores <- [85, 90, 78]
APPEND(scores, 92)
FOR EACH score IN scores
{
DISPLAY(score)
}
```

Remember: AP CSP lists use 1-based indexing!

Common Mistakes to Avoid

1. Using $=$ instead of $\leftarrow$ for assignment
2. Forgetting that lists start at index 1, not 0
3. Writing lowercase (if, else, display) instead of uppercase
4. Using semicolons (not needed in pseudocode)
5. Using cout or cin instead of DISPLAY or INPUT
6. Forgetting parentheses around conditions

Common Mistakes to Avoid

1. Using $=$ instead of $\leftarrow$ for assignment
2. Forgetting that lists start at index 1, not 0
3. Writing lowercase (if, else, display) instead of uppercase
4. Using semicolons (not needed in pseudocode)
5. Using cout or cin instead of DISPLAY or INPUT
6. Forgetting parentheses around conditions

Tip: When converting between C++ and pseudocode, focus on the logic, not the exact syntax!

Quick Reference Guide

C++	**AP CSP Pseudocode**
== = (assignment)	←
cout ¡¡	DISPLAY()
cin ¿¿	INPUT()
if	IF
else	ELSE
while	REPEAT UNTIL
for (fixed count)	REPEAT n TIMES
for (range)	FOR EACH
vector	list
.push_back()	APPEND()
.size()	LENGTH()

Summary

Key Takeaways:

- Pseudocode is easier to read than real code
- AP CSP pseudocode uses uppercase keywords
- Use $\leftarrow$ for assignment, not $=$
- Lists start at index 1 (not 0 like C++)
- DISPLAY replaces cout, INPUT replaces cin
- Control structures are similar but simpler
- Focus on logic, not exact syntax

Summary

Key Takeaways:

- Pseudocode is easier to read than real code
- AP CSP pseudocode uses uppercase keywords
- Use $\leftarrow$ for assignment, not $=$
- Lists start at index 1 (not 0 like C++)
- DISPLAY replaces cout, INPUT replaces cin
- Control structures are similar but simpler
- Focus on logic, not exact syntax

Next Steps: Practice converting between C++ and pseudocode!