NSBACI User Guide

Version 1.0.0

Nicolás Serrano García

2026

Contents

Chapter 1

NSBACI User Guide

Version 1.0 | The language reference

1.1 What is NSBACI?

NSBACI (Nicolás Serrano BAsic Concurrency Interpreter) is a small development environment designed to work with simple concurrency. The program consists of a simple text editor, which supports edition of one file, which one can then compile and run. NSBACI has it's own language, which follow C's syntax for almost every aspect. It also has I/O, whose syntax comes directly from C++. The program also carries a runtime environment, where you can analyze your programs step by step, just like a debugger.

NSBACI is inspired by Ben Ari's jBACI, but built from scratch in C++ with a renewed interface and some extra features like graphics drawing.

1.2 Getting Started

1.2.1 The Interface

When you open NSBACI, you'll see a clean code editor. The sidebar on the left has two main buttons:

- **Compile** — Checks your code for errors and loads it for execution
- **Run** — Launches the runtime view where the actual execution happens

The menu bar gives you access to file operations, editing shortcuts, and view options.

1.2.2 Your First Program

Let's start with something trivial to make sure everything works:

```
int x = 5;
cout << "Hello, x is: " << x << endl;
return;
```

Type this in the editor, press **Compile** (or press `Ctrl+B`), then **Run** (or `Ctrl+R`). You'll see the output in the console panel. As I mentioned, the syntax is greatly inspired in C++.

—.

1.2.3 Variables and Types

Four basic types:

```
int counter = 0;          // Integer numbers
bool isReady = true;      // true or false
char letter = 'A';        // Single character
const int MAX = 100;      // Constant (can't be changed)
```

1.2.4 Operators

```
// Arithmetic: + - * / %
int result = 10 + 5 * 2;   // result is 20 (not 30!)

// Comparison: < > <= >= == !=
bool bigger = (10 > 5);    // true

// Logical: && || !
bool both = true && false; // false

// Increment/Decrement
x++;                       // Same as x = x + 1
x--;                       // Same as x = x - 1

// Compound assignment
x += 5;                    // Same as x = x + 5
```

1.2.5 Arrays

Fixed-size arrays, declared with a constant size:

```
int numbers[5];           // Array of 5 integers
numbers[0] = 10;          // First element (indexing starts at 0)
numbers[4] = 50;          // Last element

int values[3] = {1, 2, 3}; // Initialize directly (if supported)
```

1.2.6 Control Flow

Standard constructs:

```
// If-else
if (x > 0) {
```

```
    cout << "Positive" << endl;
} else if (x < 0) {
    cout << "Negative" << endl;
} else {
    cout << "Zero" << endl;
}

// While loop
while (x < 10) {
    x++;
}

// Do-while (runs at least once)
do {
    x--;
} while (x > 0);

// For loop
for (int i = 0; i < 5; i++) {
    cout << i << endl;
}

// Break and continue work as expected
while (true) {
    if (condition) break;      // Exit loop
    if (other) continue;       // Skip to next iteration
}
```

1.2.7 Functions

Define functions before using them:

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

void sayHello() {
    cout << "Hello!" << endl;
    return;
}

// Using them
int sum = add(3, 4);  // sum is 7
sayHello();           // Prints "Hello!"
```

Functions can return values (`int`, `bool`, `char`) or nothing (`void`). The `return;` at the end of void functions is optional but helps with clarity.

1.2.8 Input and Output

Simple stream-based I/O:

```
// Output
cout << "The answer is: " << 42 << endl;
cout << "Multiple " << "values " << "work" << endl;

// Input
int userInput;
cin >> userInput;
```

When `cin` is called, the runtime pauses and waits for you to type a value in the input area.

1.3 Concurrency

1.3.1 Creating Parallel Blocks with cobegin/coend

The `cobegin/coend` construct creates concurrent threads:

```
cout << "Before parallel execution" << endl;

cobegin
{
    // Thread 1
    cout << "Thread 1 says hi" << endl;
}
{
    // Thread 2
    cout << "Thread 2 says hi" << endl;
}
{
    // Thread 3
    cout << "Thread 3 says hi" << endl;
}
coend

cout << "After parallel execution" << endl;
```

Each block inside `cobegin/coend` runs as a separate thread. The program continues past `coend` only when **all** threads have finished.

1.3.2 The Runtime View

When you run a concurrent program, NSBACI switches to the Runtime View. You will see the following:

- **Threads Panel** — Lists all threads with their current state (running, blocked, terminated). Click on a thread to see what instruction it's about to execute.

- **Variables Panel** — Shows all global variables and their current values. This updates in real-time as threads modify shared state.
- **Console** — Program output appears here. Input prompts also show up here.
- **Control Buttons:**
  - **Step** — Execute one instruction from a randomly chosen runnable thread
  - **Run** — Keep executing until the program ends or you pause
  - **Pause** — Stop continuous execution
  - **Reset** — Start over from the beginning
  - **Stop** — Exit the runtime and go back to editing

1.3.3 Observing Race Conditions

Here's a classic example of why concurrency is tricky:

```
int counter = 0;

cobegin
{
    int i = 0;
    while (i < 100) {
        counter++;
        i++;
    }
}
{
    int j = 0;
    while (j < 100) {
        counter++;
        j++;
    }
}
coend

cout << "Final counter: " << counter << endl;
```

You might expect `counter` to be 200. Run it a few times. Sometimes it is. Sometimes it's less. That's a **race condition** — both threads read and write `counter` without coordination, and updates get lost.

Use the Step button to go slowly and watch exactly how the interleaving causes problems.

1.4 Semaphores

Semaphores are synchronization primitives that let threads coordinate safely.

1.4.1 Declaration and Operations

```
semaphore mutex = 1;      // Binary semaphore (like a lock)
semaphore slots = 5;      // Counting semaphore

p(mutex);   // Wait: if value > 0, decrement and continue; else block
// ... critical section ...
v(mutex);   // Signal: increment value, wake a blocked thread if any
```

1.4.2 Fixing the Race Condition

```
int counter = 0;
semaphore mutex = 1;  // Only one thread at a time

cobegin
{
    int i = 0;
    while (i < 100) {
        p(mutex);         // Acquire lock
        counter++;        // Safe now
        v(mutex);         // Release lock
        i++;
    }
}
{
    int j = 0;
    while (j < 100) {
        p(mutex);
        counter++;
        v(mutex);
        j++;
    }
}
coend

cout << "Final counter: " << counter << endl;  // Always 200
```

Now `counter` is always 200. The semaphore ensures only one thread modifies `counter` at a time.

1.5 Drawing Graphics

NSBACI includes a canvas for drawing.

1.5.1 How Drawing Works

The approach is SDL-like: set the color first, then draw shapes. There are outline versions (`draw*`) and filled versions (`fill*`).

1.5.2 Setting Colors

```
// RGB values (0-255)
setColor(255, 0, 0);          // Red
setColor(0, 255, 0, 128);     // Semi-transparent green (RGBA)

// Or use predefined macros
setColor(RED);
setColor(BLUE);
setColor(GREEN);
setColor(WHITE);
setColor(BLACK);
setColor(YELLOW);
setColor(CYAN);
setColor(MAGENTA);
setColor(ORANGE);
setColor(PINK);
setColor(PURPLE);
setColor(GRAY);
```

1.5.3 Drawing Shapes

```
// Outlines
drawCircle(100, 100, 50);              // x, y, radius
drawRect(10, 10, 100, 50);             // x, y, width, height
drawLine(0, 0, 100, 100);              // x1, y1, x2, y2
drawTriangle(100, 50, 50, 150, 150, 150);  // Three points
drawEllipse(200, 200, 60, 30);         // x, y, radiusX, radiusY
drawPixel(50, 50);                     // Single point

// Filled shapes
fillCircle(200, 200, 30);
fillRect(150, 10, 80, 40);
fillTriangle(200, 50, 150, 150, 250, 150);
fillEllipse(300, 200, 40, 20);

// Text
drawText(10, 20, "Hello World!");
drawText(10, 50, "Big text", 24);      // With font size
```

1.5.4 Using Points

You can also specify coordinates as `{x, y}` points:

```
drawCircle({100, 100}, 50);            // Point + radius
fillRect({10, 10}, 100, 50);           // Point + dimensions
drawLine({0, 0}, {100, 100});          // Two points
drawTriangle({100, 50}, {50, 150}, {150, 150});  // Three points
```

1.5.5 Position Macros

For convenience, predefined positions (based on 800×600 canvas):

```
drawCircle(CENTER, 50);          // Center of canvas
drawCircle(TOP_LEFT, 30);        // Top-left corner
drawCircle(BOTTOM_RIGHT, 30);    // Bottom-right corner
// Also: TOP_CENTER, TOP_RIGHT, CENTER_LEFT, CENTER_RIGHT,
//       BOTTOM_LEFT, BOTTOM_CENTER
```

1.5.6 Canvas Operations

```
clearCanvas();                    // Clear to white
clearCanvas(0, 0, 0);             // Clear to black
clearCanvas(BLUE);                // Clear to blue

setLineWidth(3);                  // Thicker lines

refreshCanvas();                  // Force display update
```

1.5.7 Example: A Simple House

```
clearCanvas(135, 206, 235);      // Sky blue

// House body
setColor(139, 69, 19);           // Brown
fillRect(200, 250, 200, 150);

// Roof
setColor(RED);
fillTriangle({200, 250}, {300, 150}, {400, 250});

// Door
setColor(101, 67, 33);
fillRect({275, 310}, 50, 90);

// Windows
setColor(CYAN);
fillRect({220, 280}, 40, 40);
fillRect({340, 280}, 40, 40);

// Sun
setColor(YELLOW);
fillCircle({650, 80}, 50);

// Ground
setColor(34, 139, 34);           // Forest green
fillRect({0, 400}, 800, 200);
```

```
refreshCanvas();
return;
```

1.6 Keyboard Shortcuts

Action	Shortcut
New file	`Ctrl+N`
Open file	`Ctrl+O`
Save	`Ctrl+S`
Save As	`Ctrl+Shift+S`
Compile	`Ctrl+B`
Run	`Ctrl+R`
Undo	`Ctrl+Z`
Redo	`Ctrl+Shift+Z`
Cut	`Ctrl+X`
Copy	`Ctrl+C`
Paste	`Ctrl+V`
Select All	`Ctrl+A`
Toggle Sidebar	`Ctrl+\`
Fullscreen	`F11`

NSBACI — Learning concurrency