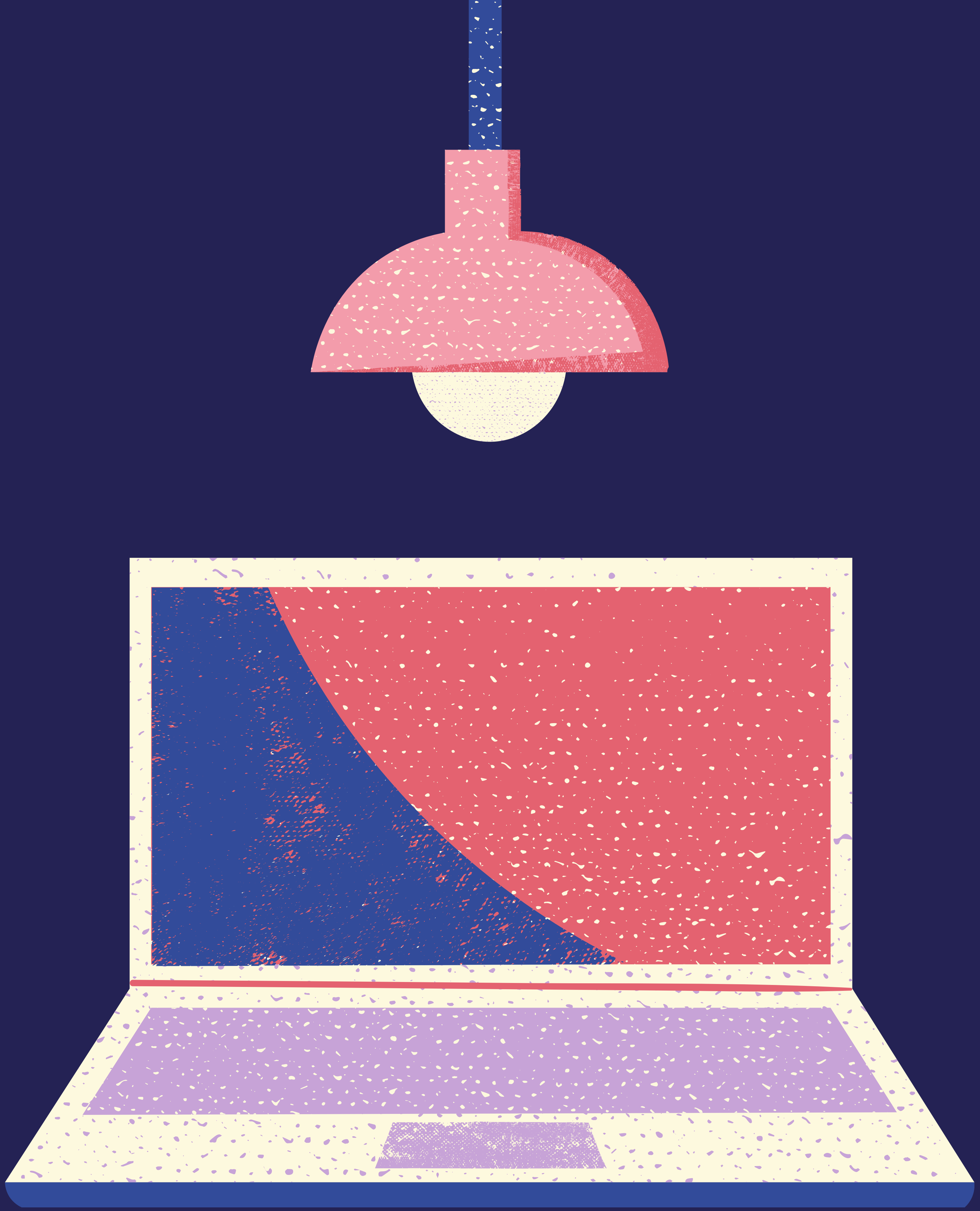


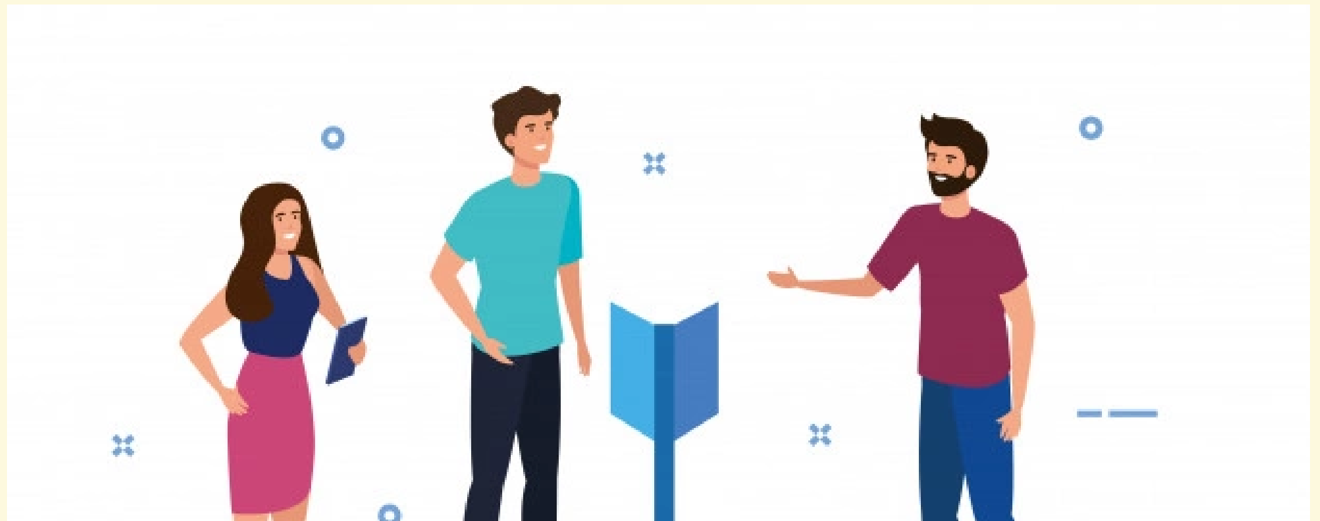
ALPHAX compiler

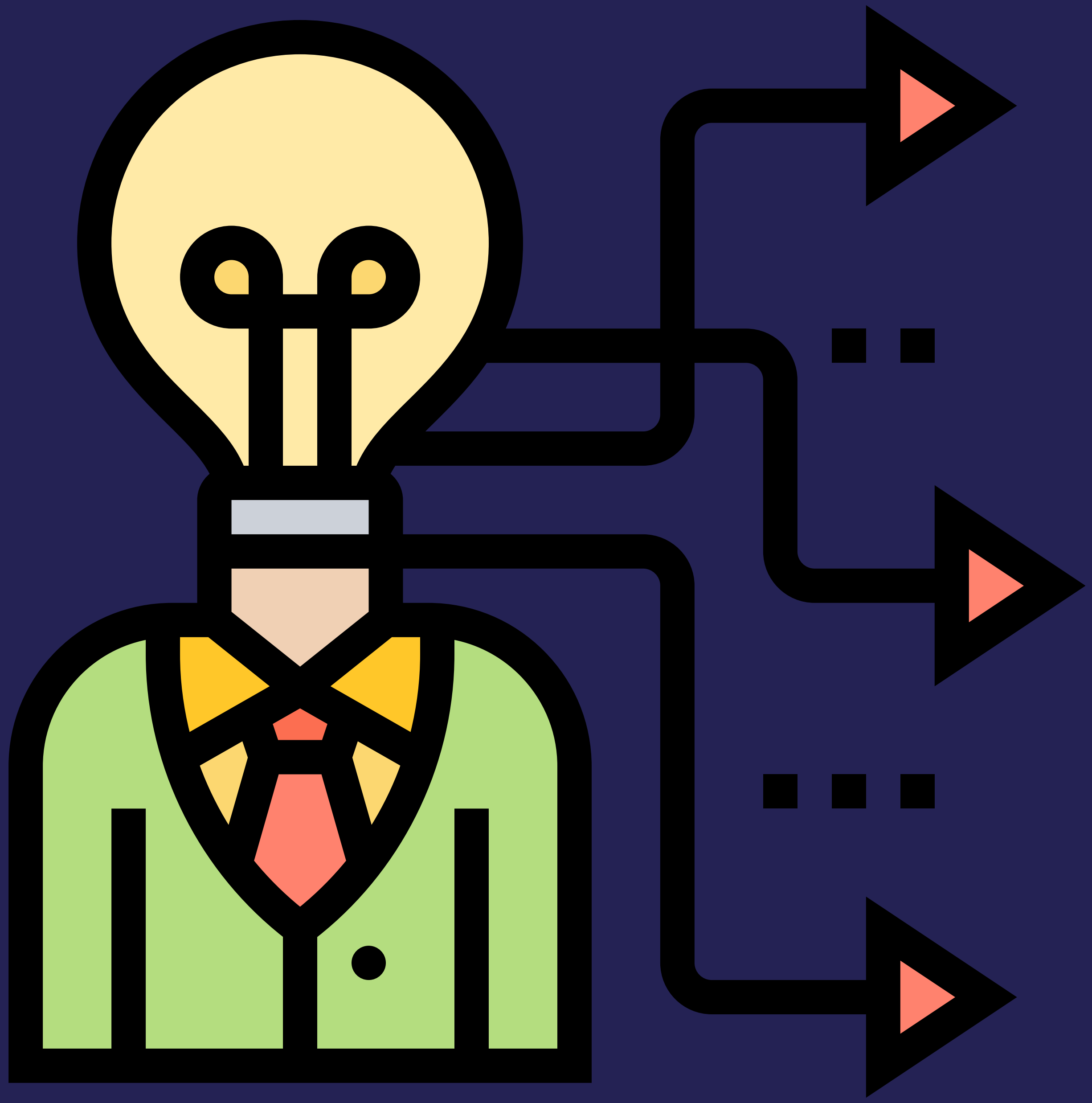
Fourth delivery

DEVELOPERS

ALPHAX Team

- Flores Constantino Diego
- Rojas Castañeda Karen Arleth



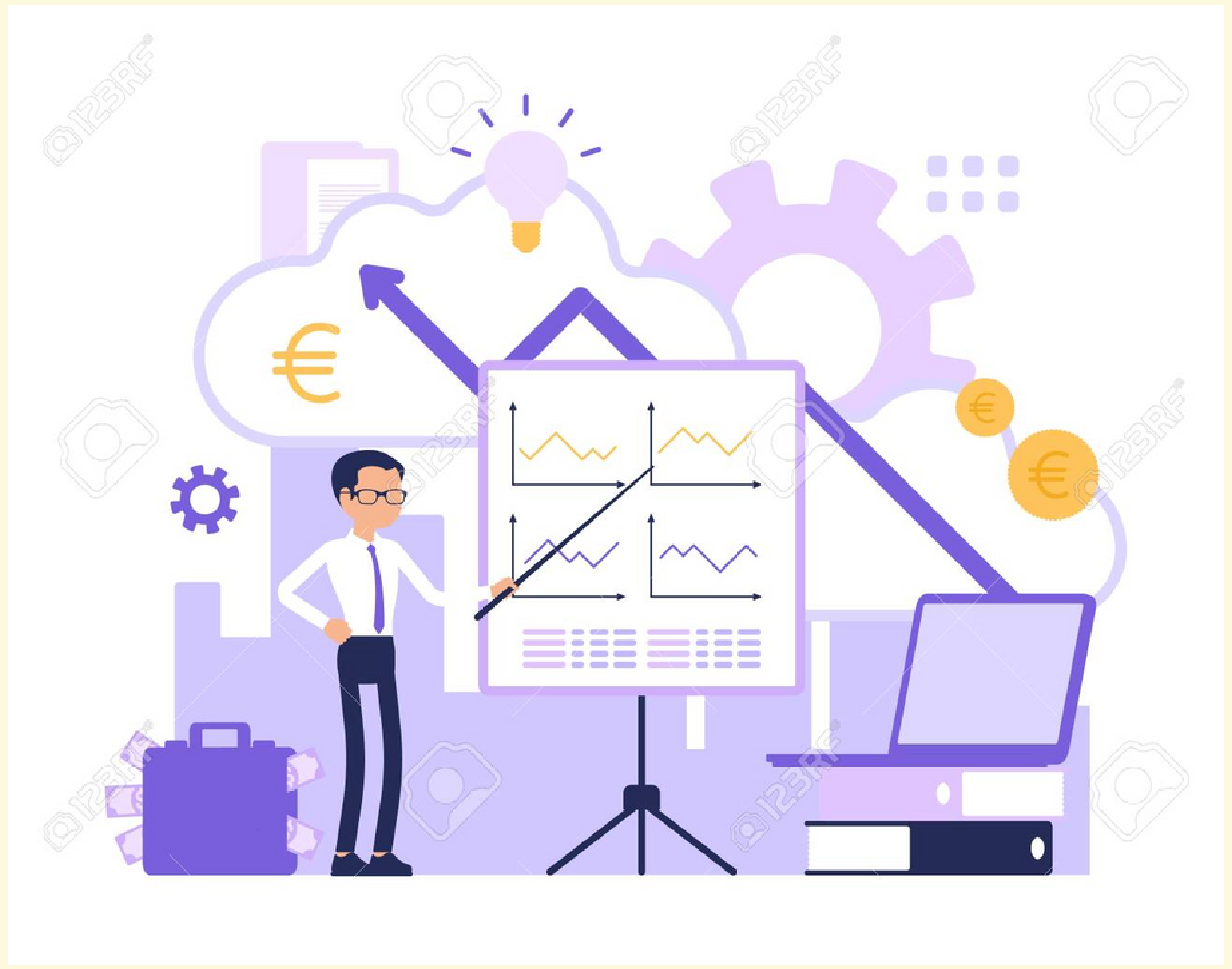


Changes.

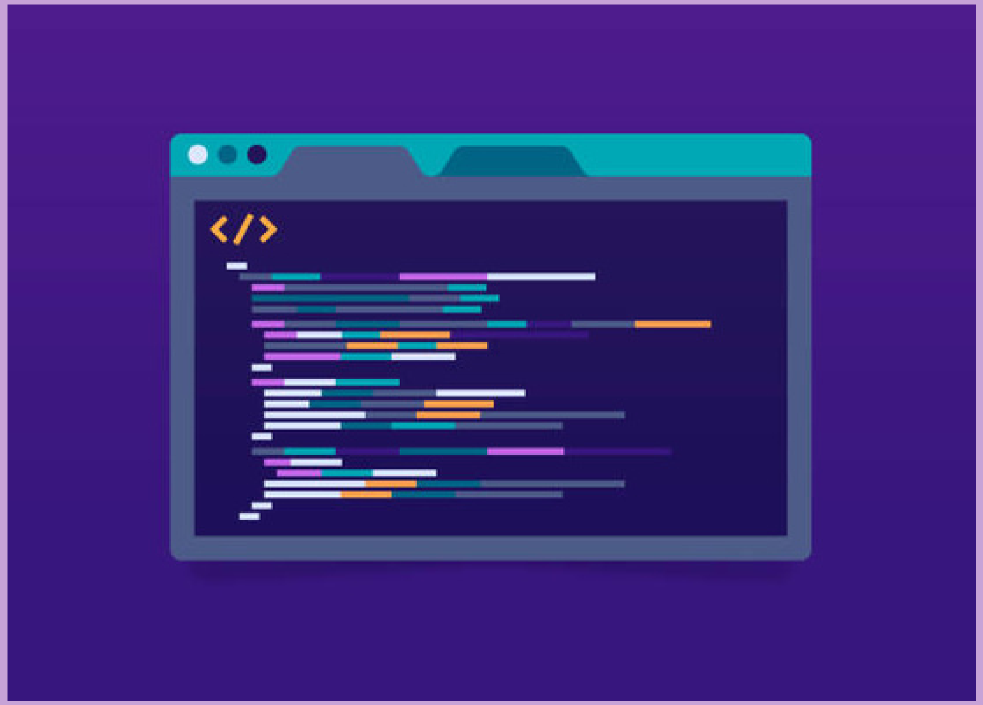
Eight operators were implemented

- Logical AND (&&)
- Logical OR (| |)
- Equal to (==)
- Not equal to (!=)
- Less than (<)
- Less than or equal to (<=)
- Greater than (>)
- Greater than or equal (>=)

AlphaX Compiler

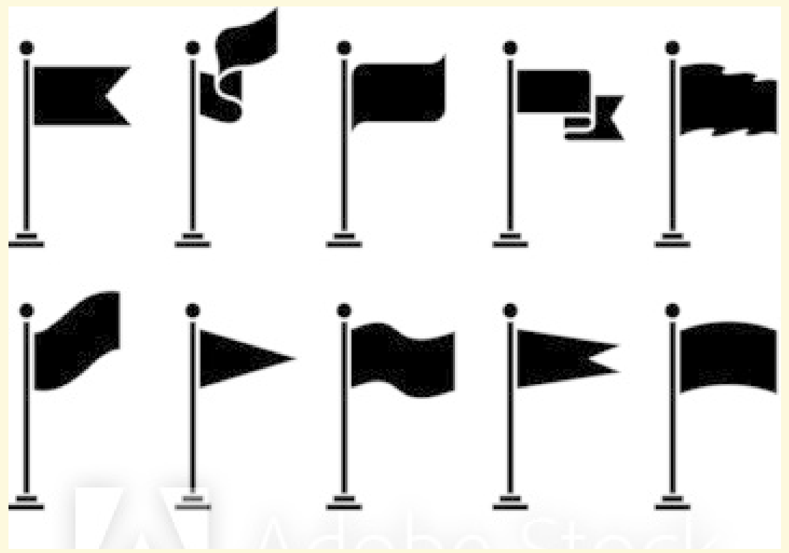


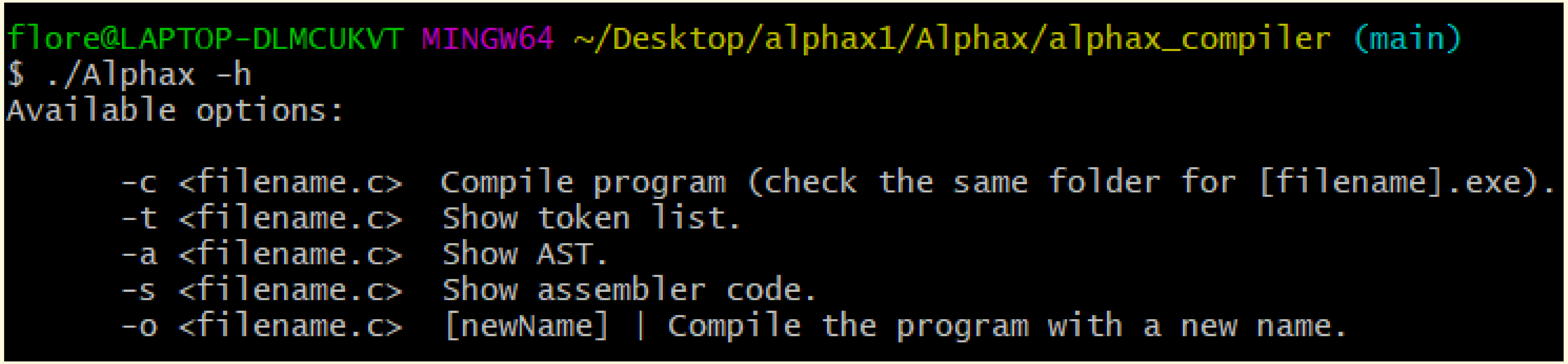
alphaX Digital Services



Implementation

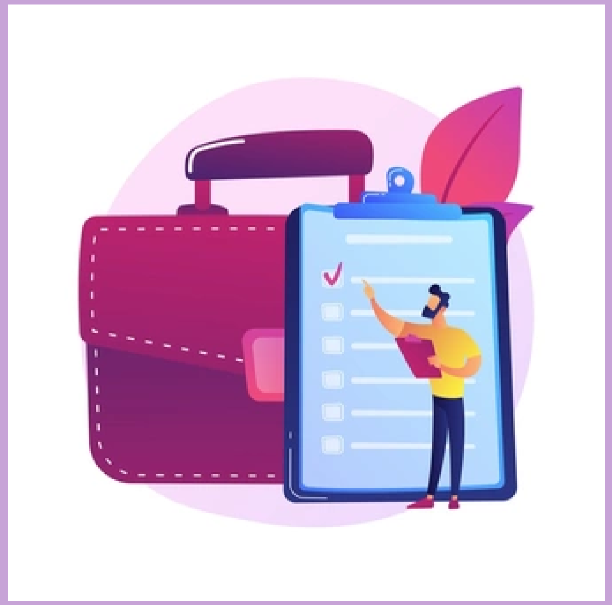
Flags input

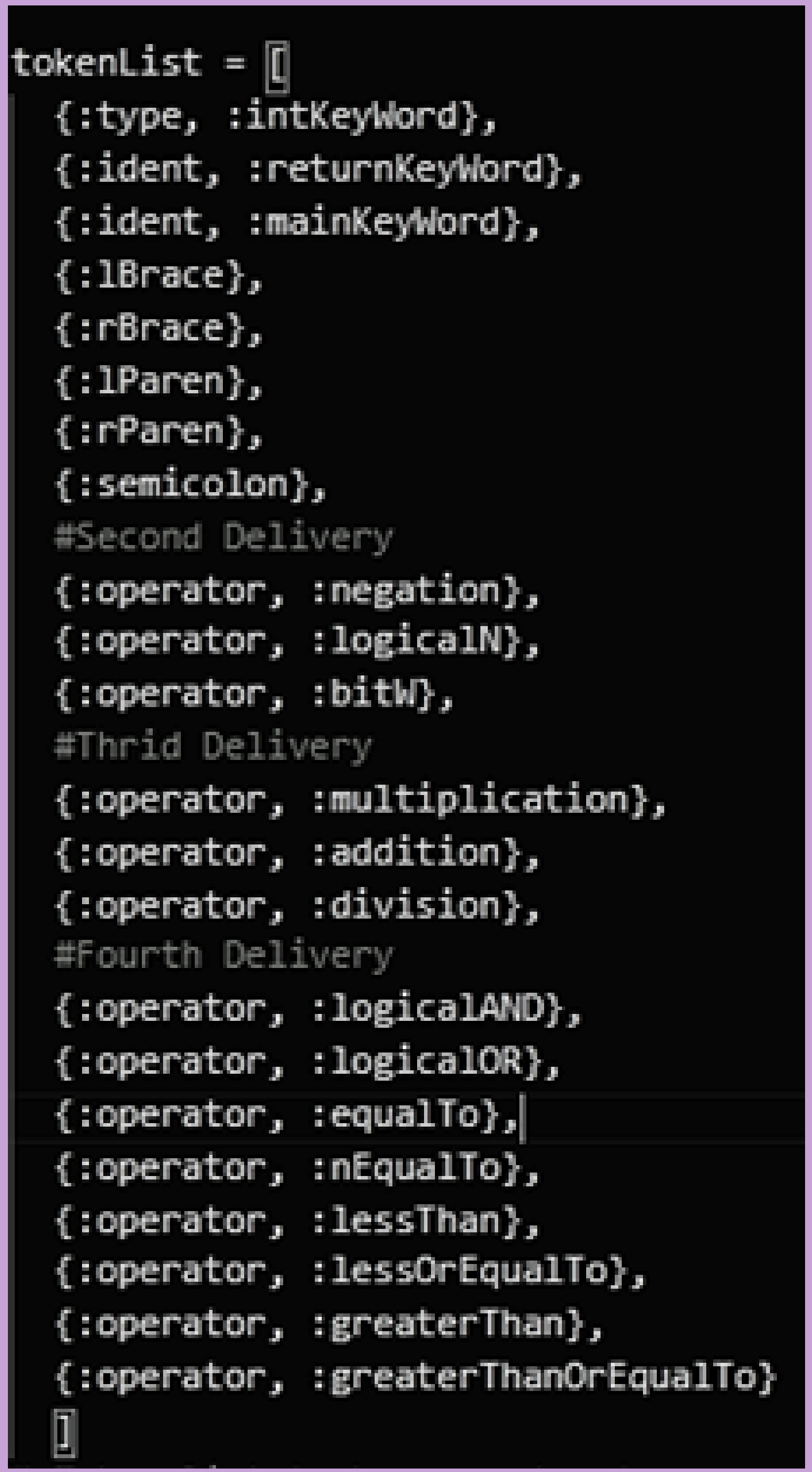




```
flore@LAPTOP-DLMCUKVT MINGW64 ~/Desktop/alphax1/Alphax/alphax_compiler (main)
$ ./Alphax -h
Available options:

      -c <filename.c>  Compile program (check the same folder for [filename].exe).
      -t <filename.c>  Show token list.
      -a <filename.c>  Show AST.
      -s <filename.c>  Show assembler code.
      -o <filename.c>  [newName] | Compile the program with a new name.
```

Token List



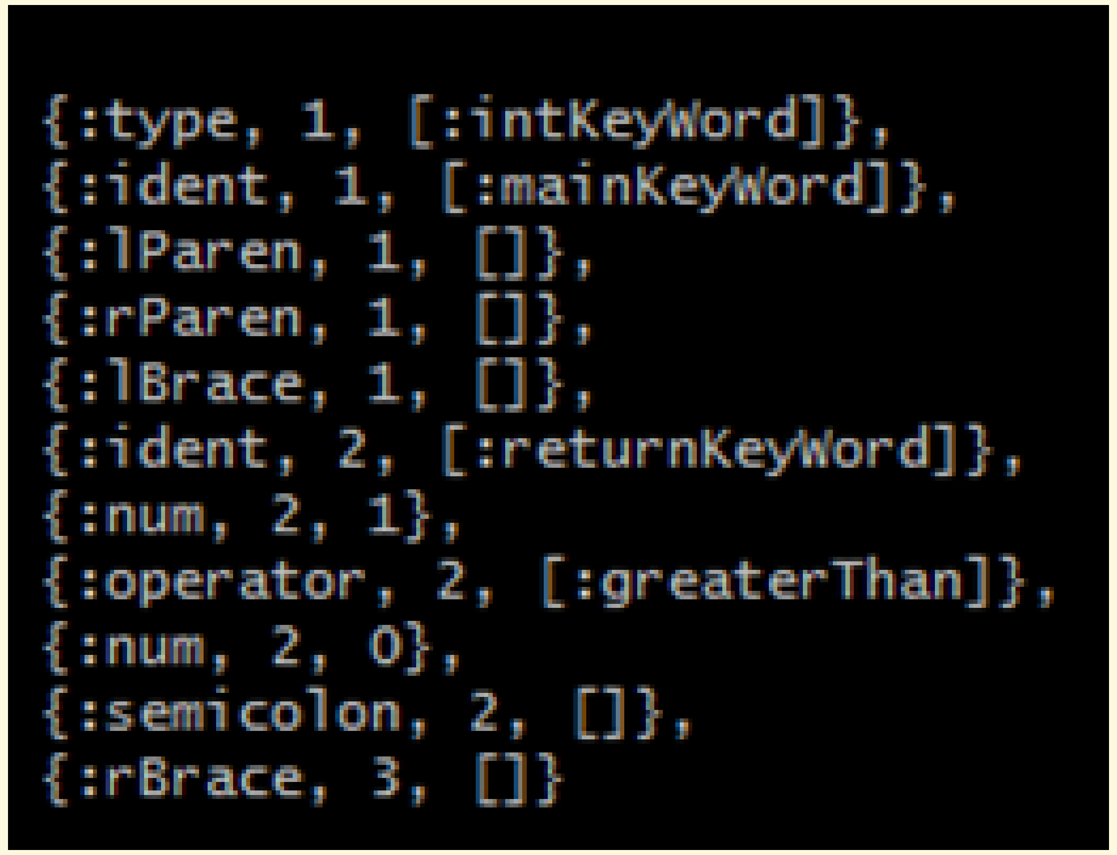


```
tokenList = [
  {:type, :intKeyWord},
  {:ident, :returnKeyWord},
  {:ident, :mainKeyWord},
  {:lBrace},
  {:rBrace},
  {:lParen},
  {:rParen},
  {:semicolon},
  #Second Delivery
  {:operator, :negation},
  {:operator, :logicalN},
  {:operator, :bitW},
  #Thrid Delivery
  {:operator, :multiplication},
  {:operator, :addition},
  {:operator, :division},
  #Fourth Delivery
  {:operator, :logicalAND},
  {:operator, :logicalOR},
  {:operator, :equalTo},
  {:operator, :nEqualTo},
  {:operator, :lessThan},
  {:operator, :lessOrEqualTo},
  {:operator, :greaterThan},
  {:operator, :greaterThanOrEqualTo}
  ]
```

$\left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \\ \end{array}\right\}$ New tokens

Token List

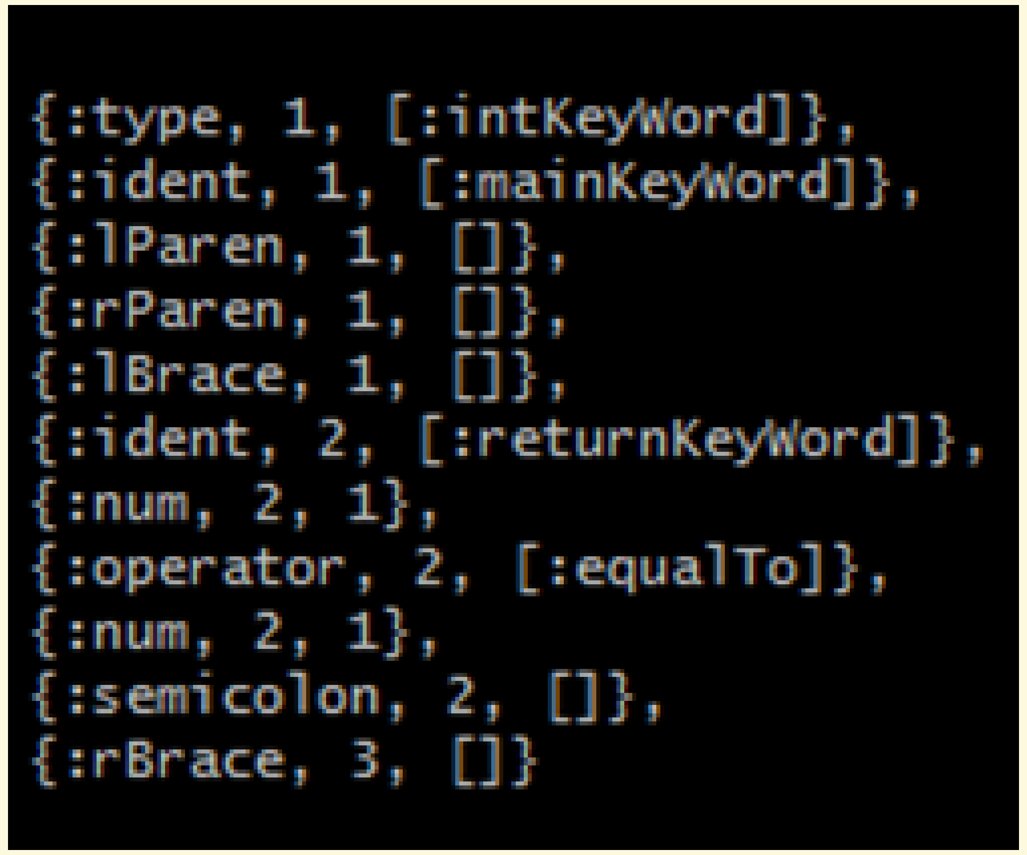
Greater than (>)



```
{:type, 1, [:intKeyWord]},
{:ident, 1, [:mainKeyWord]},
{:lParen, 1, []},
{:rParen, 1, []},
{:lBrace, 1, []},
{:ident, 2, [:returnKeyWord]},
{:num, 2, 1},
{:operator, 2, [:greaterThan]},
{:num, 2, 0},
{:semicolon, 2, []},
{:rBrace, 3, []}
```

$\left\{ \text{New token} \right\}$

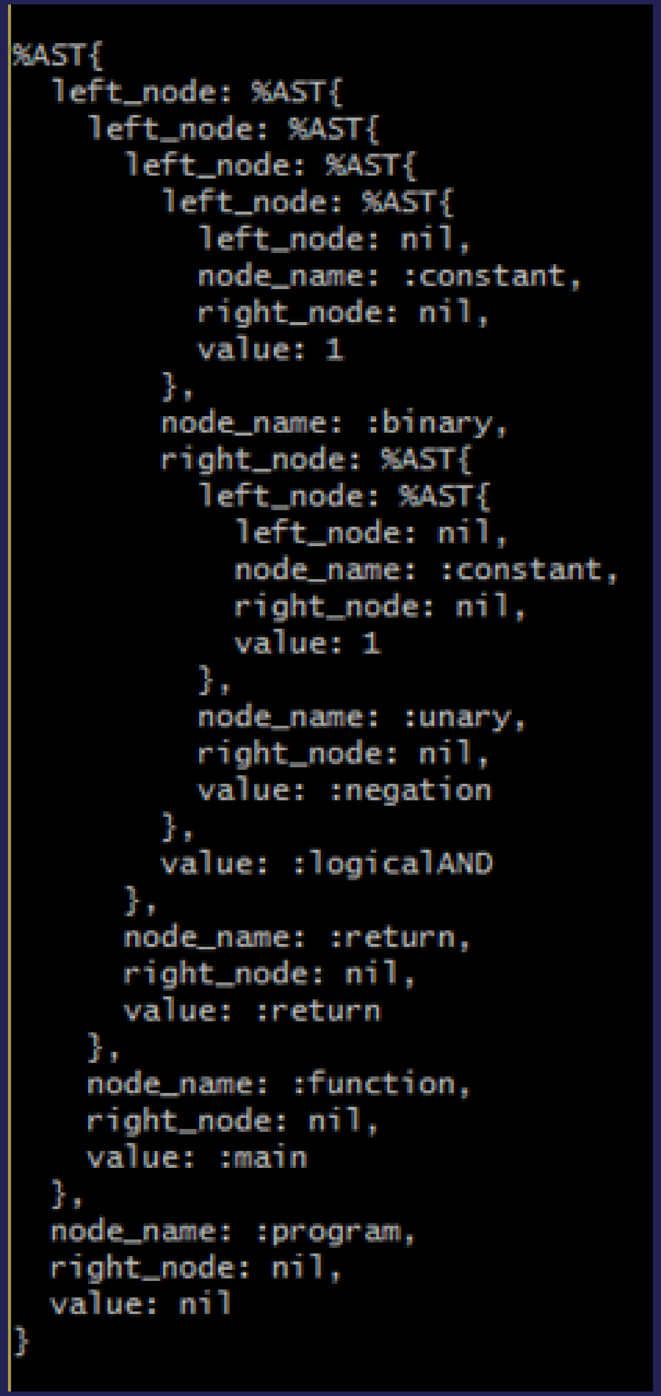
Equal to (==)



```
{:type, 1, [:intKeyWord]},
{:ident, 1, [:mainKeyWord]},
{:lParen, 1, []},
{:rParen, 1, []},
{:lBrace, 1, []},
{:ident, 2, [:returnKeyWord]},
{:num, 2, 1},
{:operator, 2, [:equalTo]},
{:num, 2, 1},
{:semicolon, 2, []},
{:rBrace, 3, []}
```

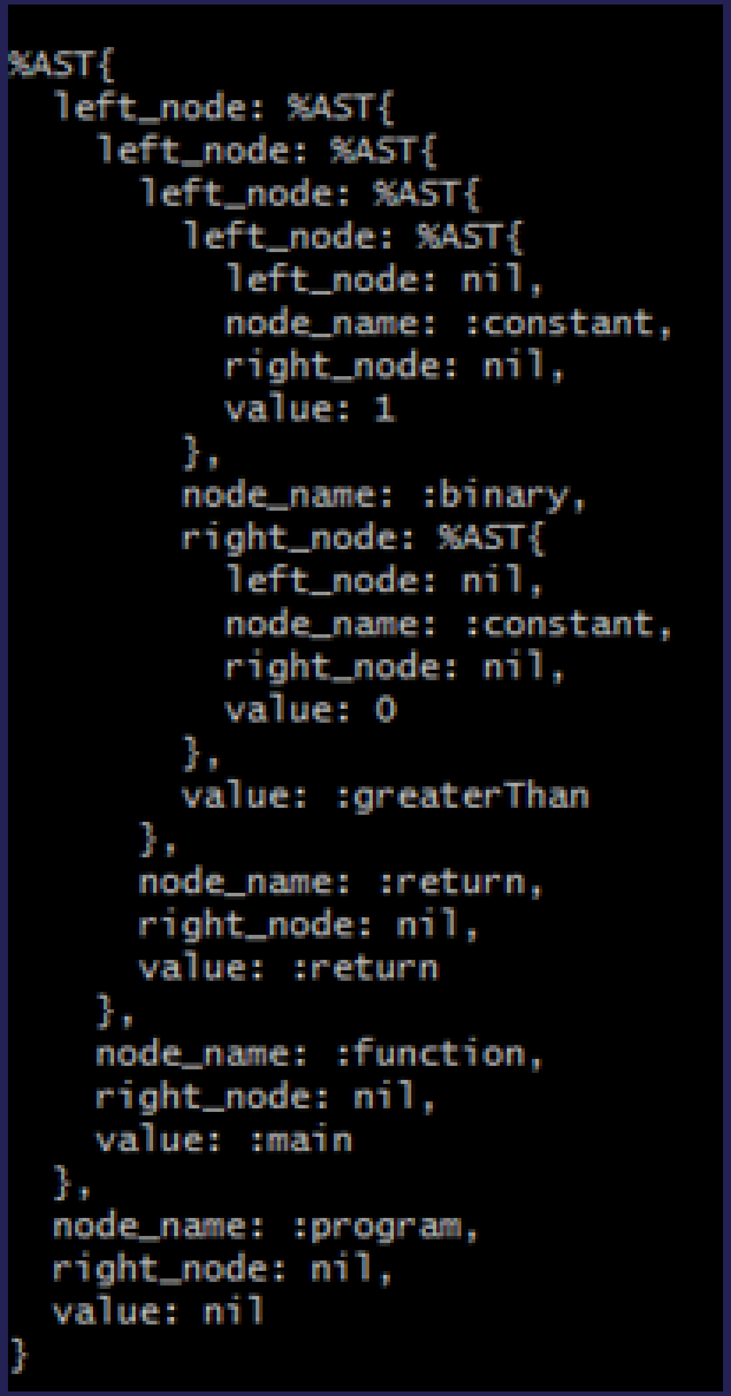
Abstract Syntax Tree

AND (&&)



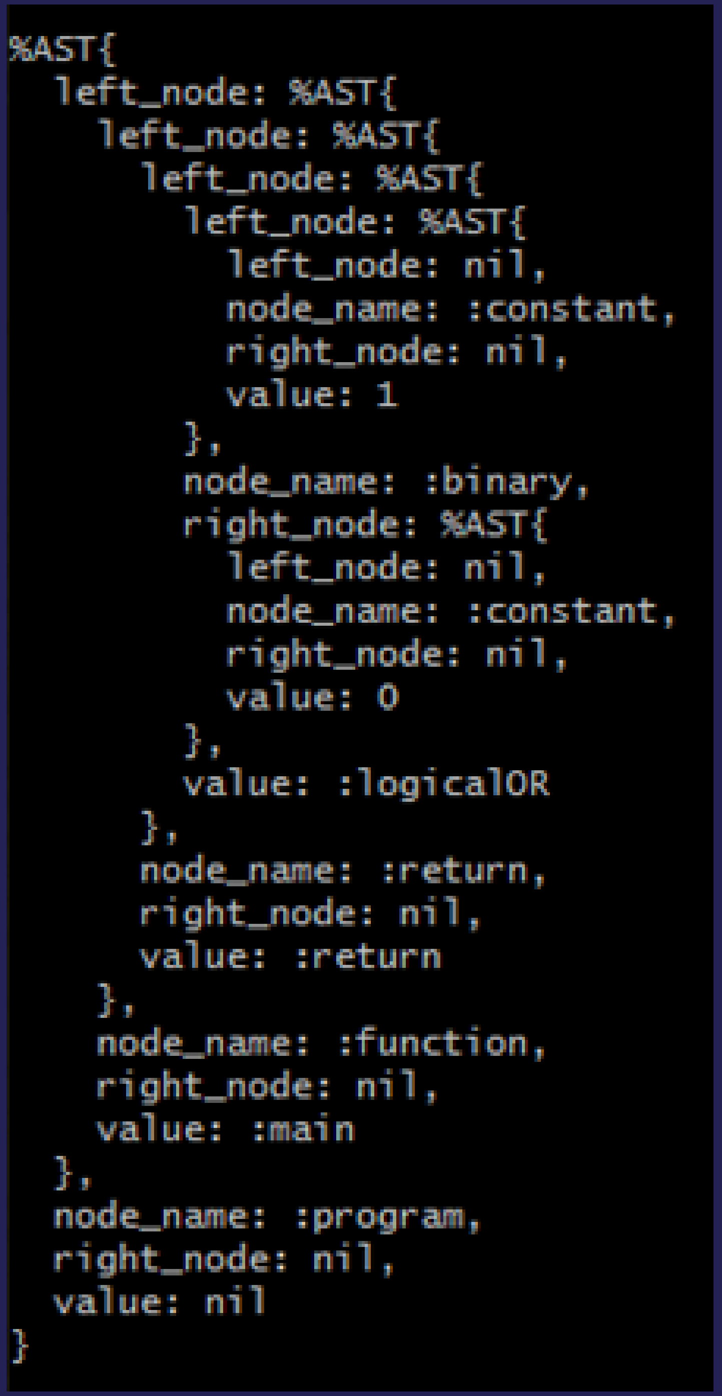
```
%AST{
  left_node: %AST{
    left_node: %AST{
      left_node: %AST{
        left_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 1
        },
        node_name: :binary,
        right_node: %AST{
          left_node: %AST{
            left_node: nil,
            node_name: :constant,
            right_node: nil,
            value: 1
          },
          node_name: :unary,
          right_node: nil,
          value: :negation
        },
        value: :logicalAND
      },
      node_name: :return,
      right_node: nil,
      value: :return
    },
    node_name: :function,
    right_node: nil,
    value: :main
  },
  node_name: :program,
  right_node: nil,
  value: nil
}
```

Greater than (>)



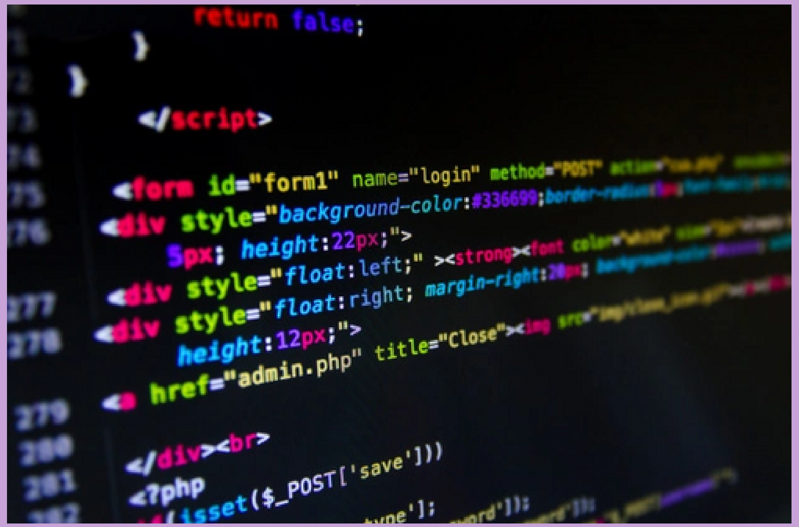
```
%AST{
  left_node: %AST{
    left_node: %AST{
      left_node: %AST{
        left_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 1
        },
        node_name: :binary,
        right_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 0
        },
        value: :greaterThan
      },
      node_name: :return,
      right_node: nil,
      value: :return
    },
    node_name: :function,
    right_node: nil,
    value: :main
  },
  node_name: :program,
  right_node: nil,
  value: nil
}
```

OR (||)

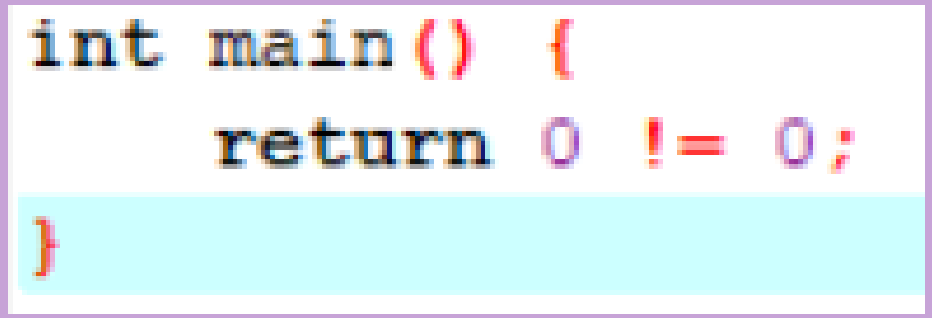


```
%AST{
  left_node: %AST{
    left_node: %AST{
      left_node: %AST{
        left_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 1
        },
        node_name: :binary,
        right_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 0
        },
        value: :logicalOR
      },
      node_name: :return,
      right_node: nil,
      value: :return
    },
    node_name: :function,
    right_node: nil,
    value: :main
  },
  node_name: :program,
  right_node: nil,
  value: nil
}
```

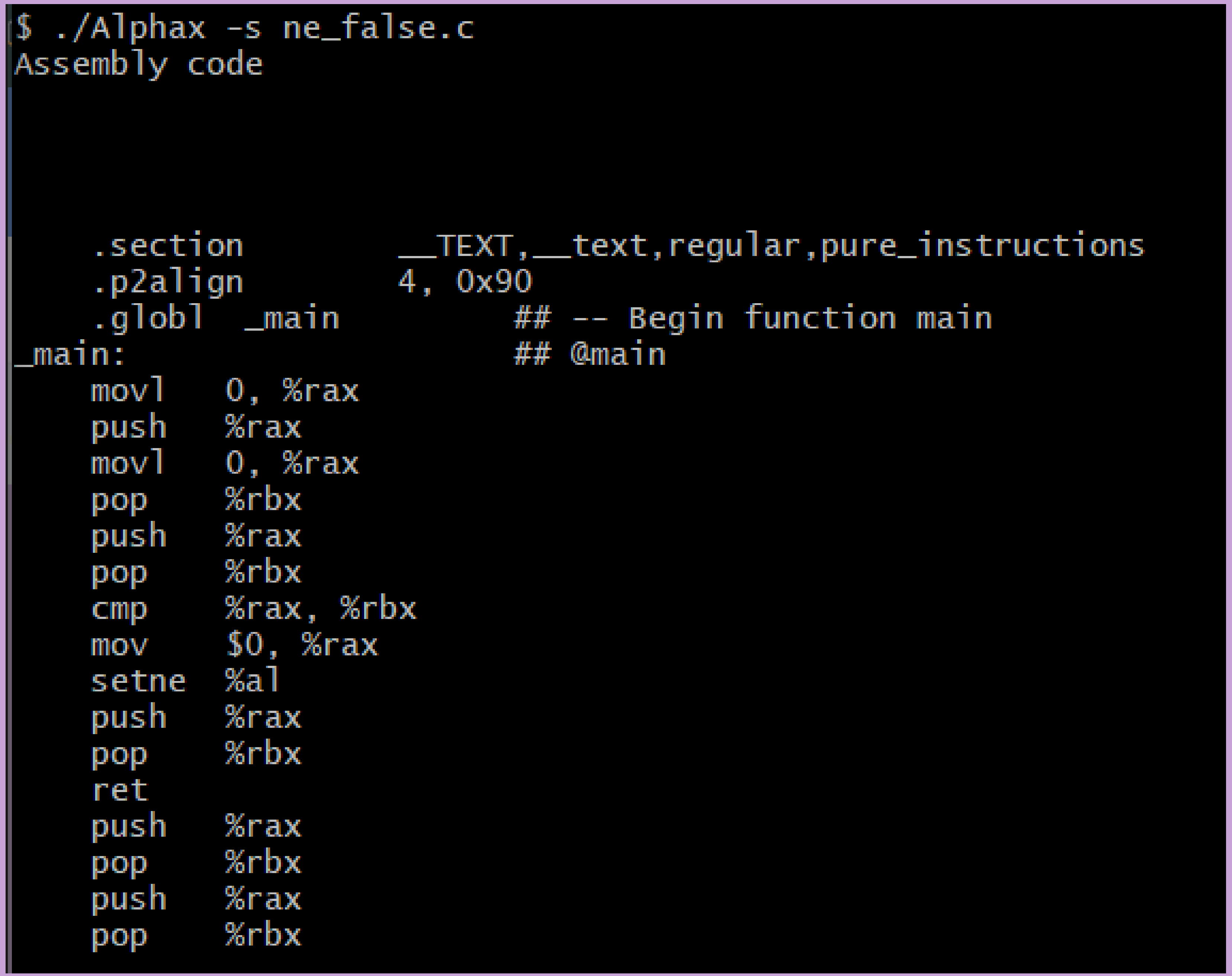
Assembler Code



Not equal To - False
(!=)



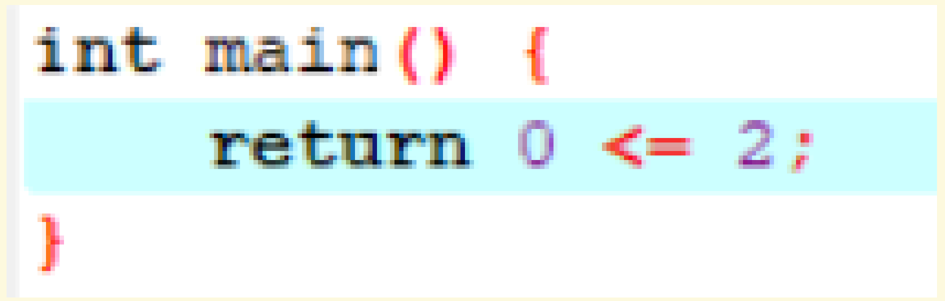
```
int main() {
    return 0 != 0;
}
```



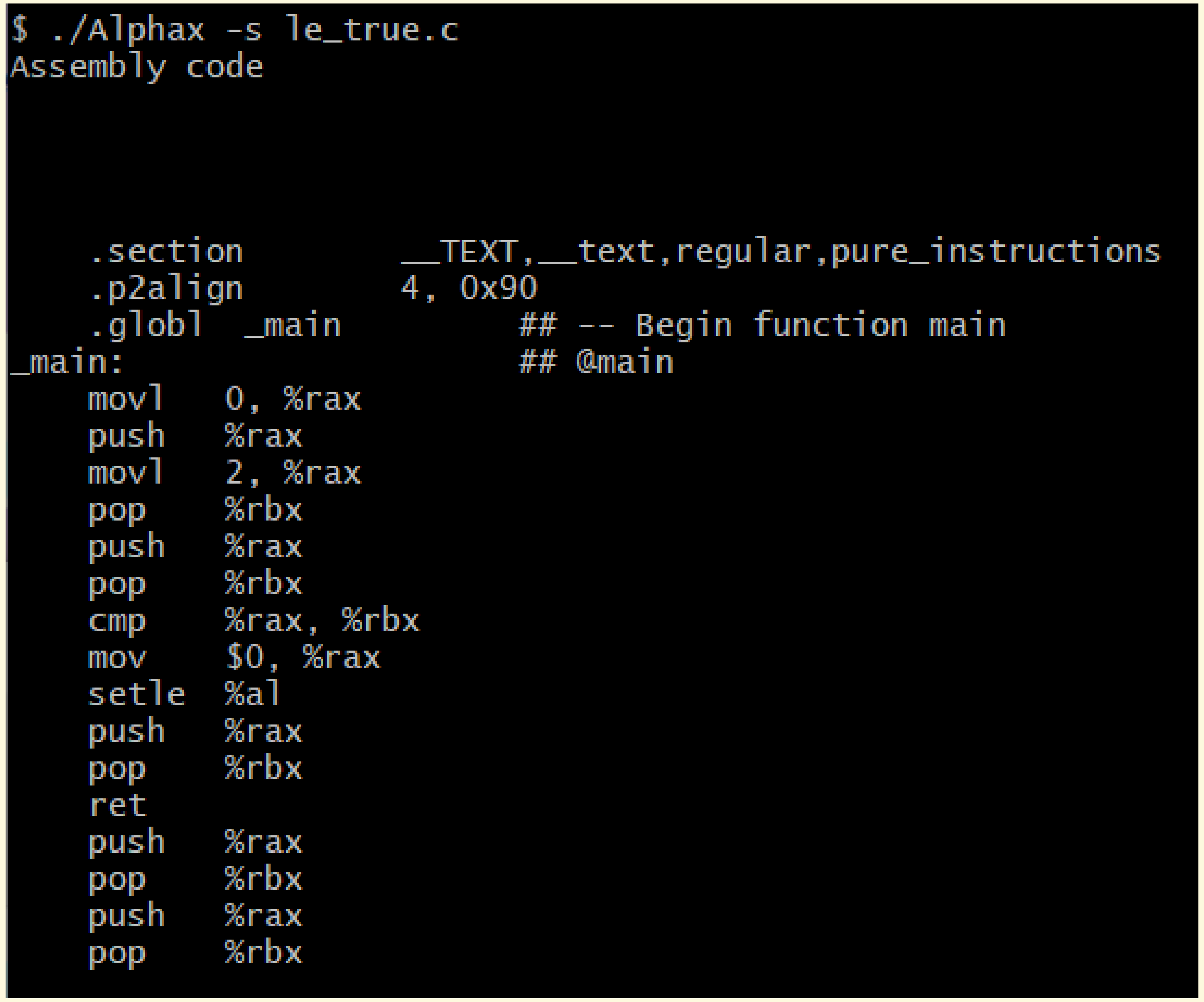
```
$ ./Alphax -s ne_false.c
Assembly code




    .section        __TEXT,__text,regular,pure_instructions
    .p2align        4, 0x90
    .globl  _main               ## -- Begin function main
_main:                          ## @main
    movl    0, %rax
    push    %rax
    movl    0, %rax
    pop     %rbx
    push    %rax
    pop     %rbx
    cmp     %rax, %rbx
    mov     $0, %rax
    setne   %al
    push    %rax
    pop     %rbx
    ret
    push    %rax
    pop     %rbx
    push    %rax
    pop     %rbx
```

Assembler Code

Less than or Equal To - True (<)



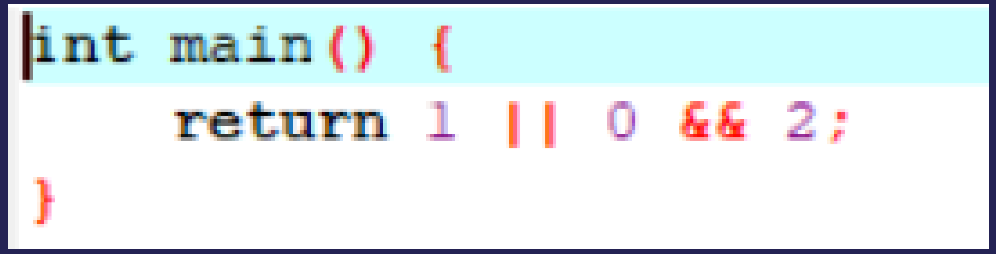
```
int main() {
    return 0 <= 2;
}
```



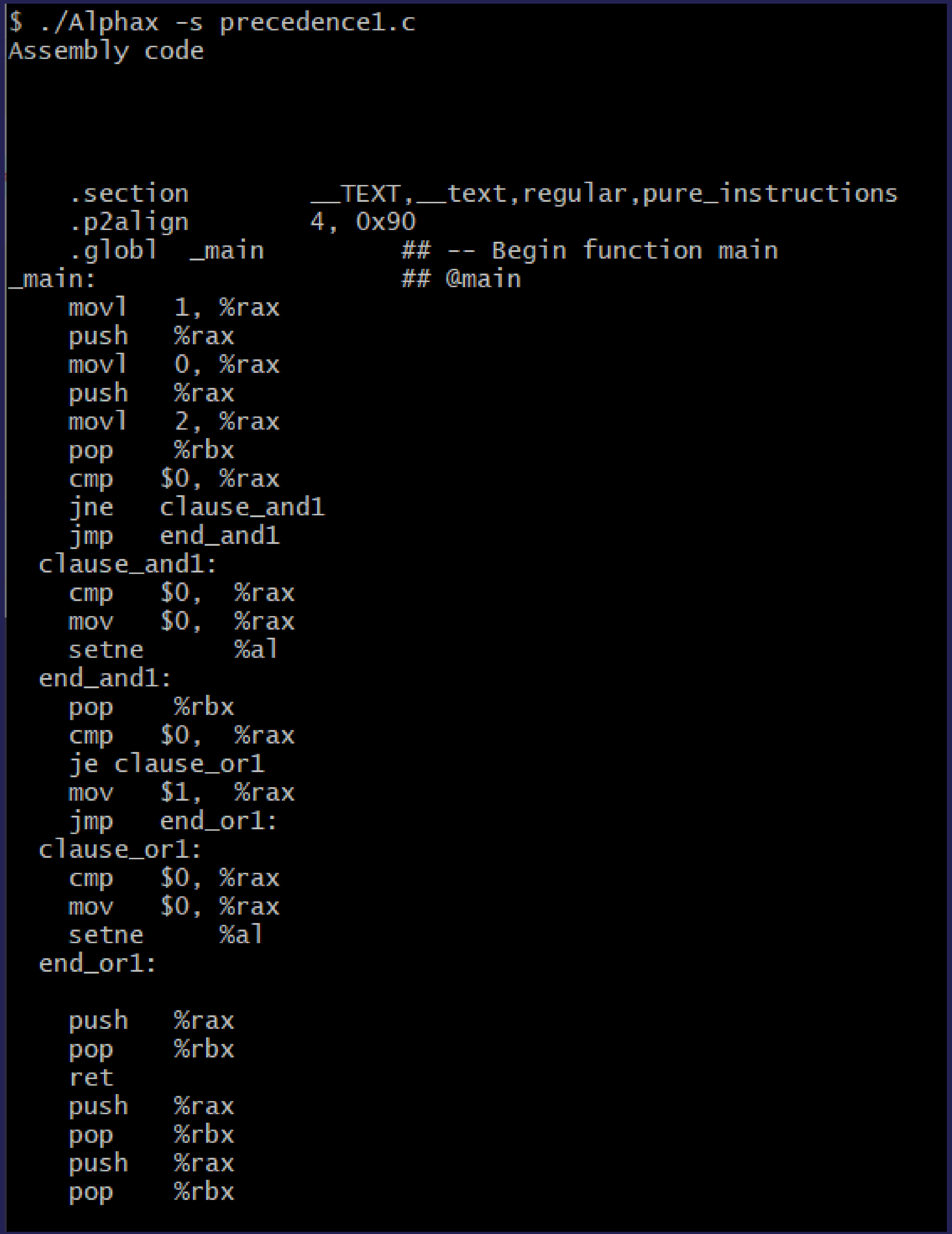
```
$ ./Alphax -s le_true.c
Assembly code




    .section        __TEXT,__text,regular,pure_instructions
    .p2align        4, 0x90
    .globl  _main               ## -- Begin function main
_main:                          ## @main
    movl    0, %rax
    push    %rax
    movl    2, %rax
    pop     %rbx
    push    %rax
    pop     %rbx
    cmp     %rax, %rbx
    mov     $0, %rax
    setle   %al
    push    %rax
    pop     %rbx
    ret
    push    %rax
    pop     %rbx
    push    %rax
    pop     %rbx
```

Assembler Code

General Example (Precedence; || and &&)

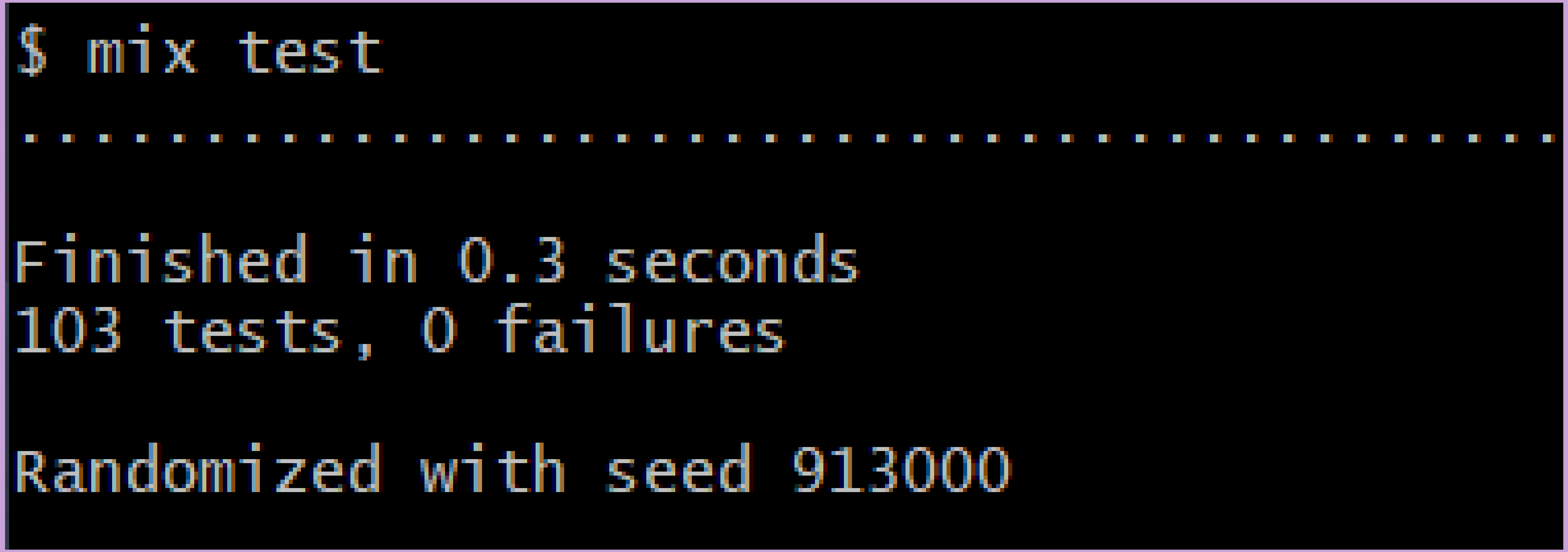


```
int main() {
    return 1 || 0 && 2;
}
```



```
$ ./Alphax -s precedence1.c
Assembly code




    .section        __TEXT,__text,regular,pure_instructions
    .p2align        4, 0x90
    .globl  _main         ## -- Begin function main
_main:                            ## @main
    movl    1, %rax
    push    %rax
    movl    0, %rax
    push    %rax
    movl    2, %rax
    pop     %rbx
    cmp    $0, %rax
    jne    clause_and1
    jmp    end_and1
  clause_and1:
    cmp    $0,  %rax
    mov    $0,  %rax
    setne       %al
  end_and1:
    pop     %rbx
    cmp    $0,  %rax
    je clause_or1
    mov    $1,  %rax
    jmp    end_or1:
  clause_or1:
    cmp    $0, %rax
    mov    $0, %rax
    setne      %al
  end_or1:

    push    %rax
    pop     %rbx
    ret
    push    %rax
    pop     %rbx
    push    %rax
    pop     %rbx
```

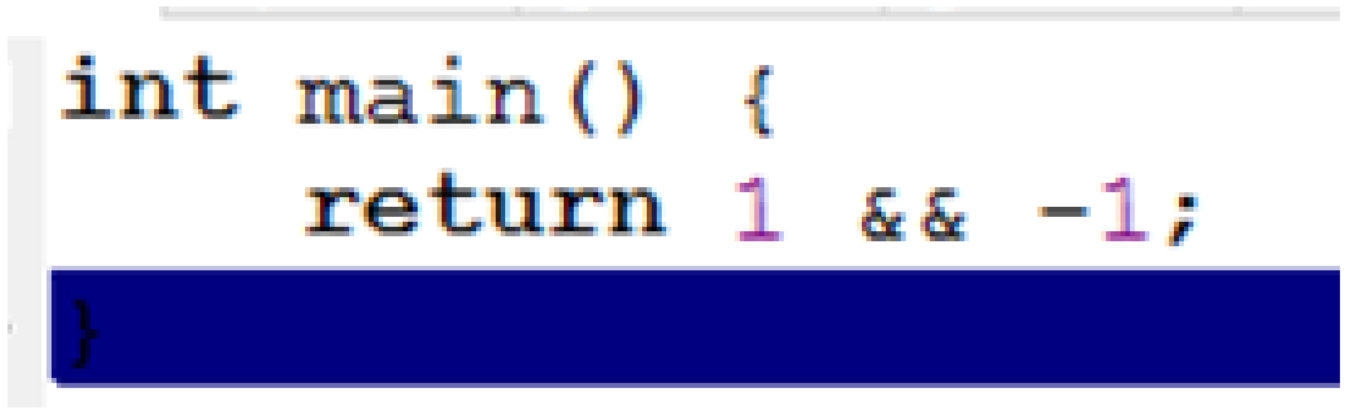
Test Plan



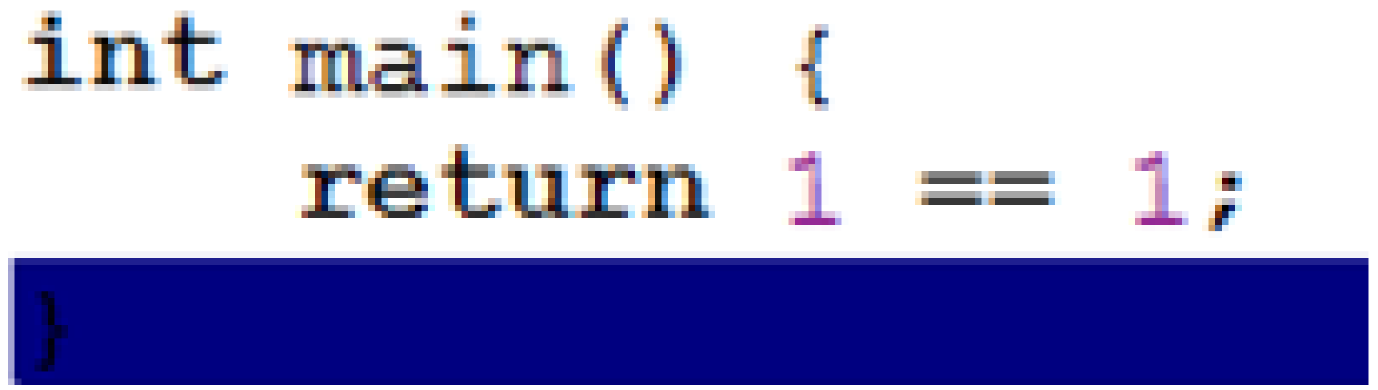
```
$ mix test
...............................................

Finished in 0.3 seconds
103 tests, 0 failures

Randomized with seed 913000
```

Test plan

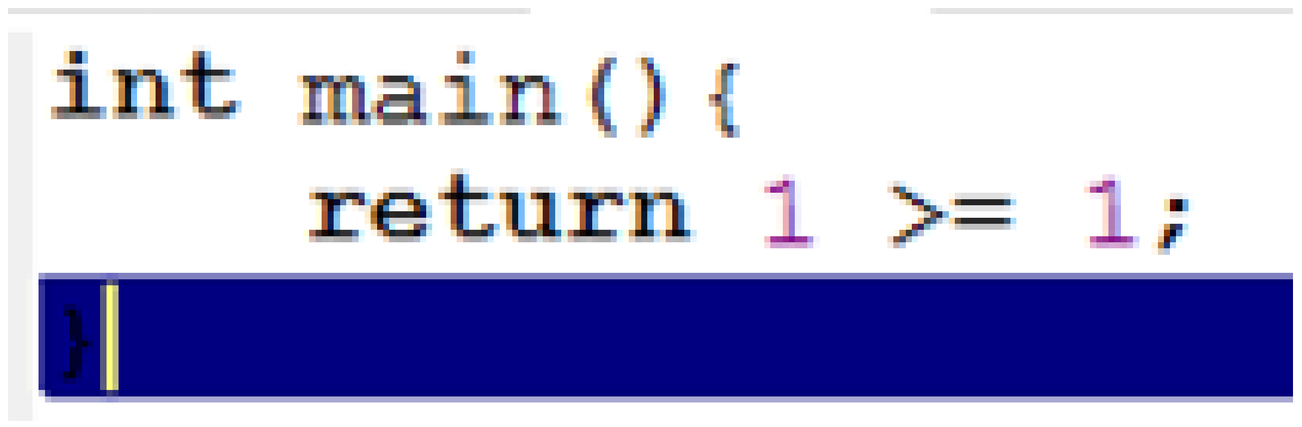
To pass



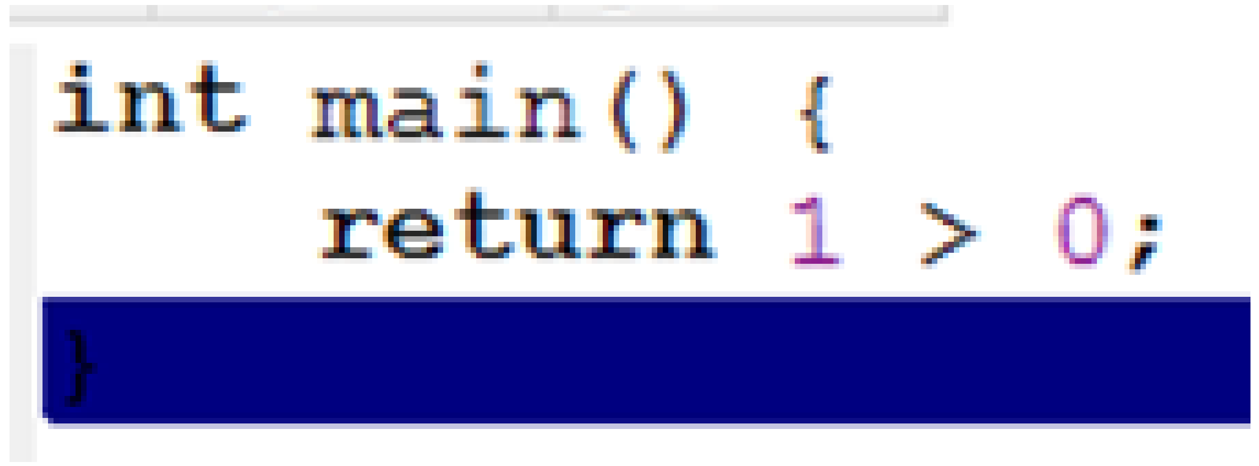
```
int main() {
    return 1 && -1;
}
```



```
int main() {
    return 1 == 1;
}
```



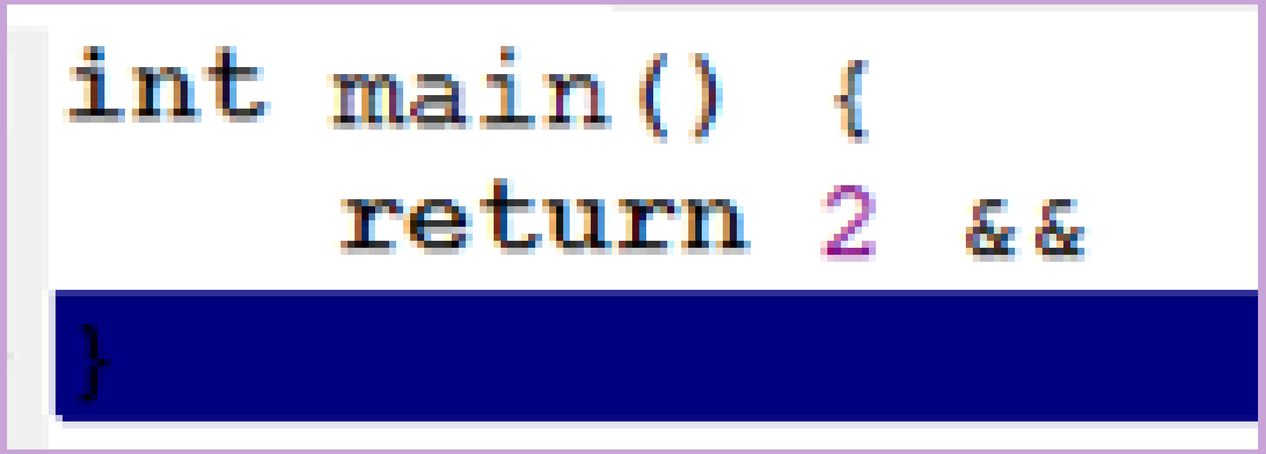
```
int main(){
    return 1 >= 1;
}
```



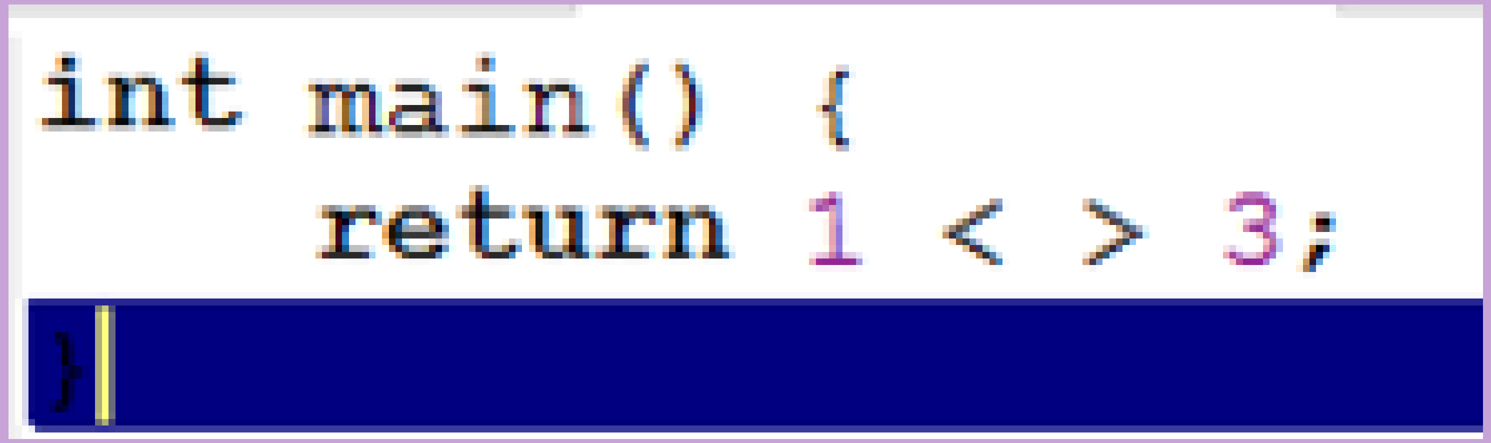
```
int main() {
    return 1 > 0;
}
```

Test plan

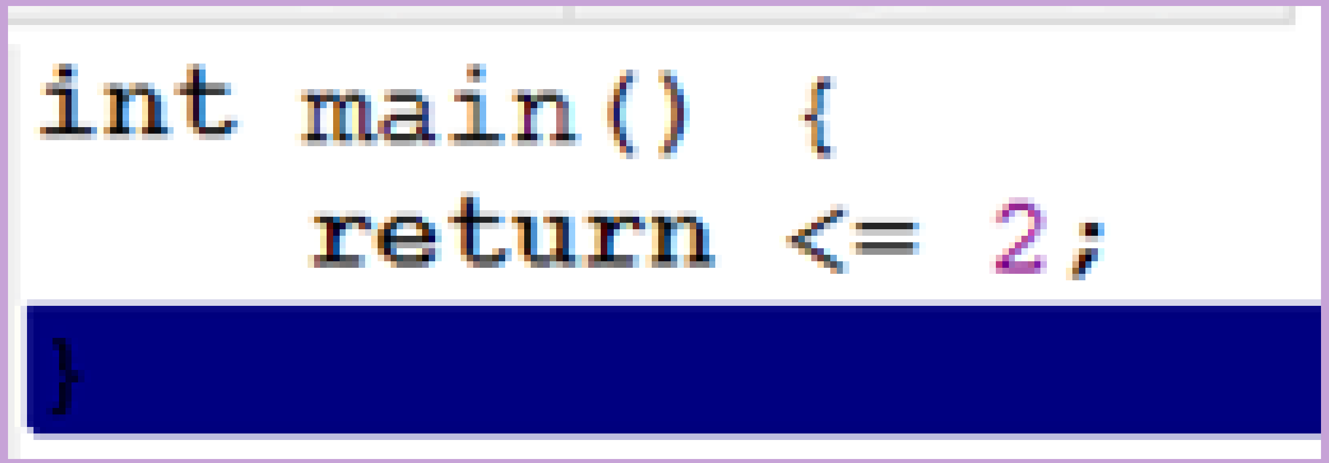
To fail



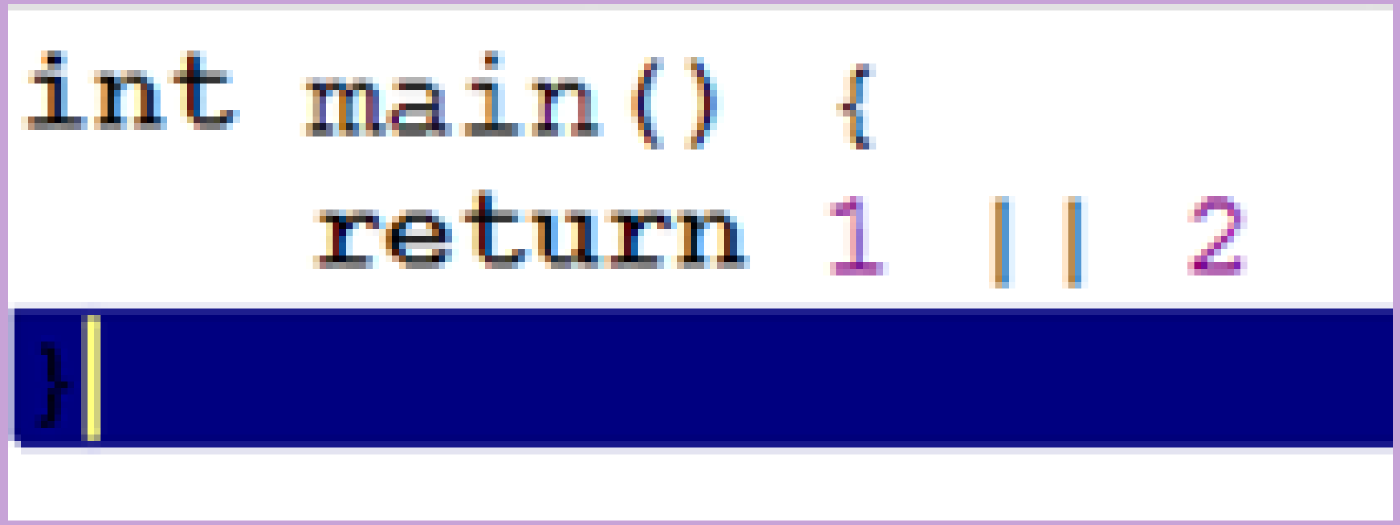
```
int main() {
    return 2 &&
}
```



```
int main() {
    return 1 < > 3;
}
```



```
int main() {
    return <= 2;
}
```



```
int main() {
    return 1 || 2
}
```

Use of github

For the version control we used a github repository

CONCLUSIONS & LEARNED LESSONS