08 - Going Loopy - Part 1

Dr. Robert Lowe

Division of Mathematics and Computer Science
Maryville College

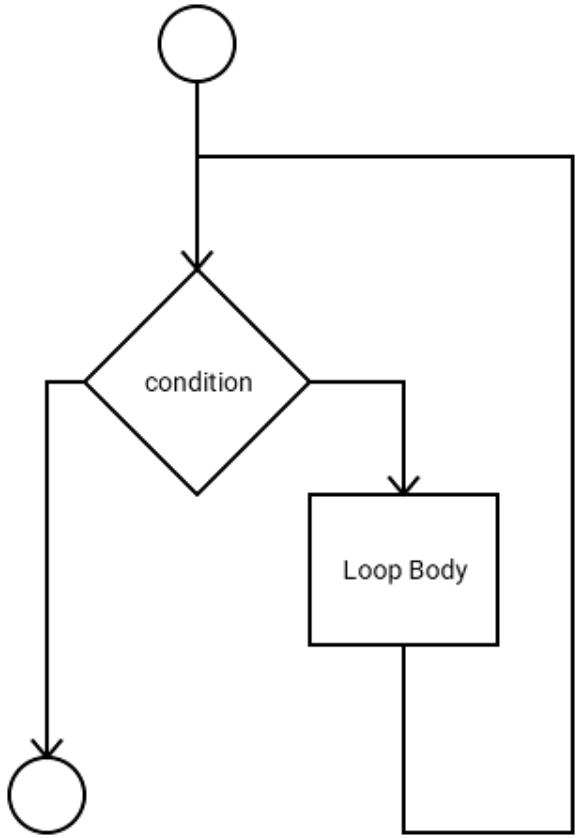
Outline

Outline

The Precondition Loop

While Loop Syntax

```
while( condition )
    statement/block
```

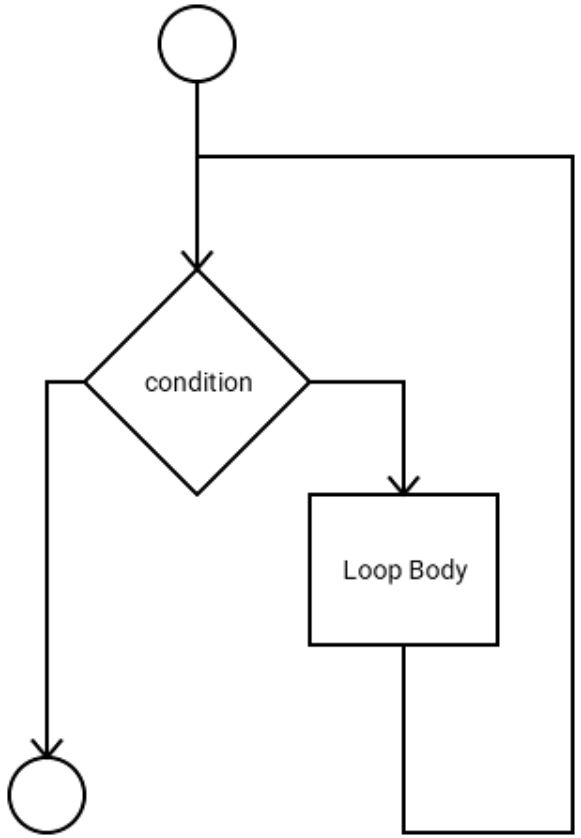


The Precondition Loop

While Loop Syntax

```
while( condition )
    statement/block
```

- The `while` loop is called a precondition loop.

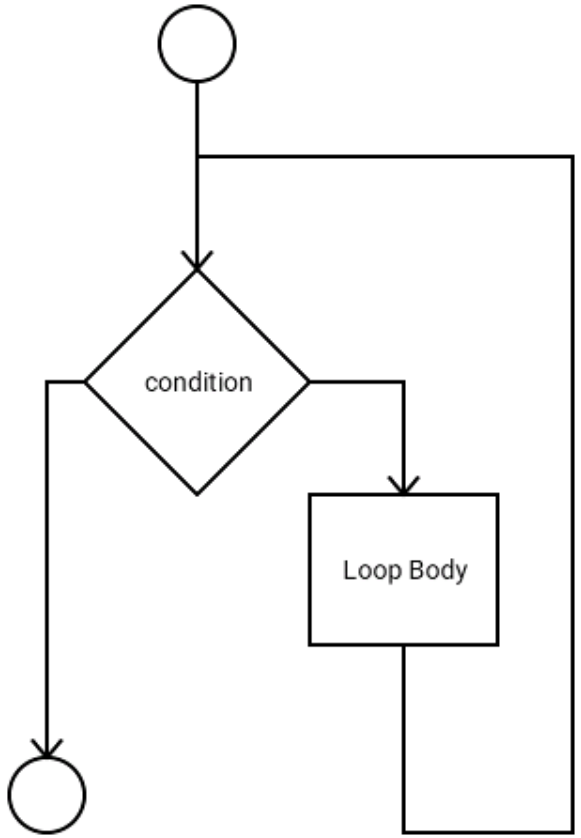


The Precondition Loop

While Loop Syntax

```
while( condition )
    statement/block
```

- The `while` loop is called a precondition loop.
- The condition is checked before the loop body is executed.

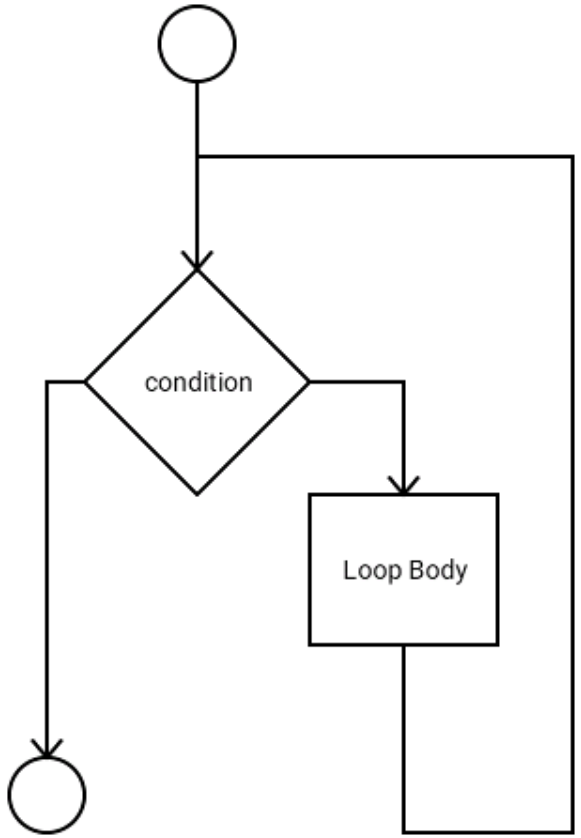


The Precondition Loop

While Loop Syntax

```
while( condition )
    statement/block
```

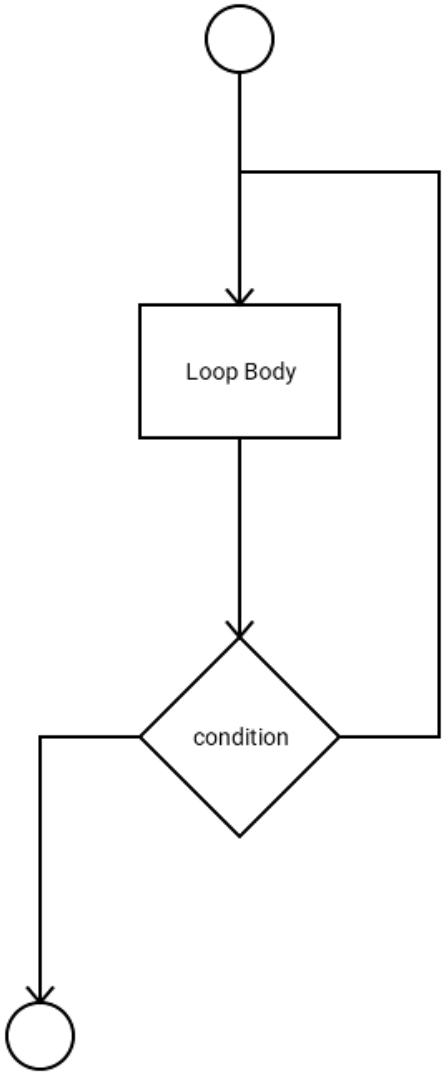
- The `while` loop is called a precondition loop.
- The condition is checked before the loop body is executed.
- The loop body is executed zero or more times.



The Postcondition Loop

While Loop Syntax

```
do
        statement/block
while( condition );
```

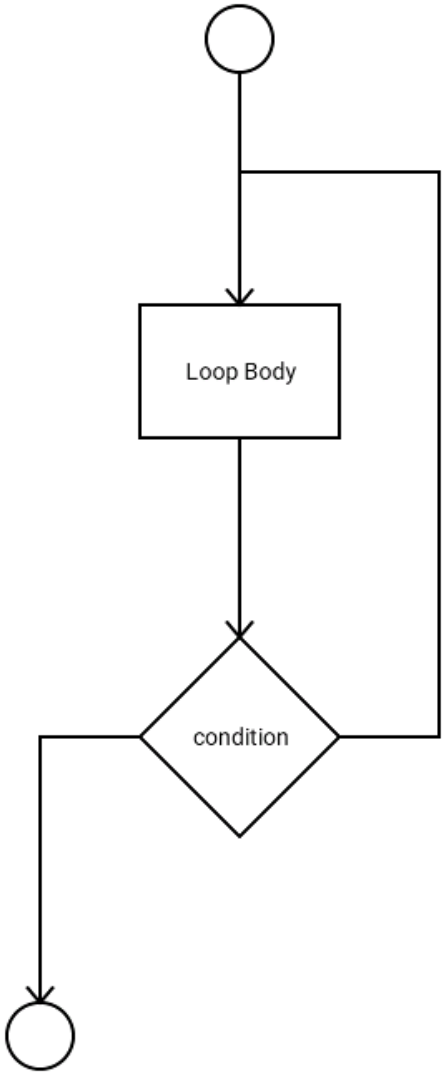


The Postcondition Loop

While Loop Syntax

```
do
        statement/block
while( condition );
```

- The `do..while` loop is called the postcondition loop.

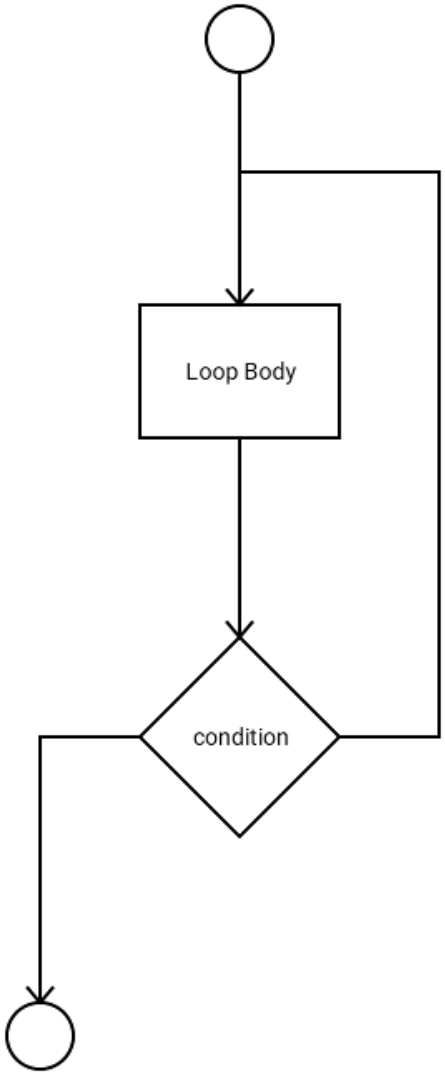


The Postcondition Loop

While Loop Syntax

```
do
        statement/block
while( condition );
```

- The `do..while` loop is called the postcondition loop.
- The condition is checked after the loop body.

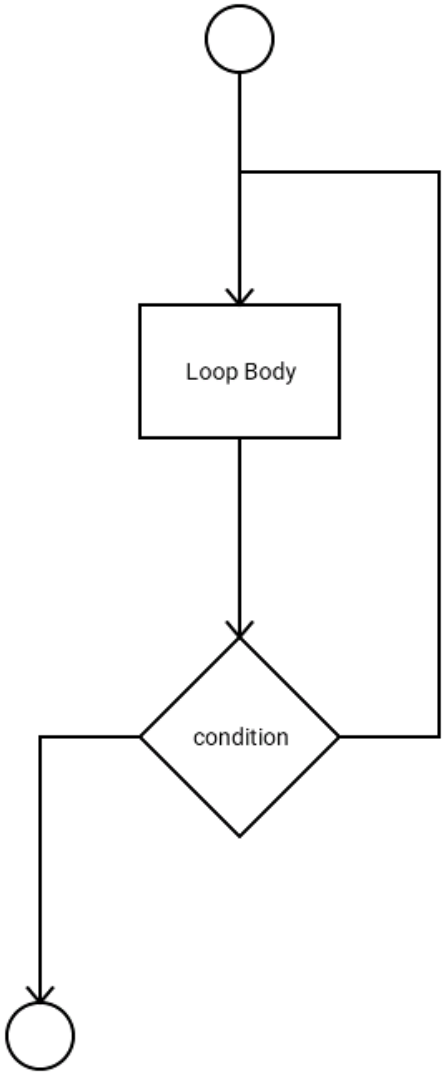


The Postcondition Loop

While Loop Syntax

```
do
        statement/block
while( condition );
```

- The `do..while` loop is called the postcondition loop.
- The condition is checked after the loop body.
- Executes 1 or more times.

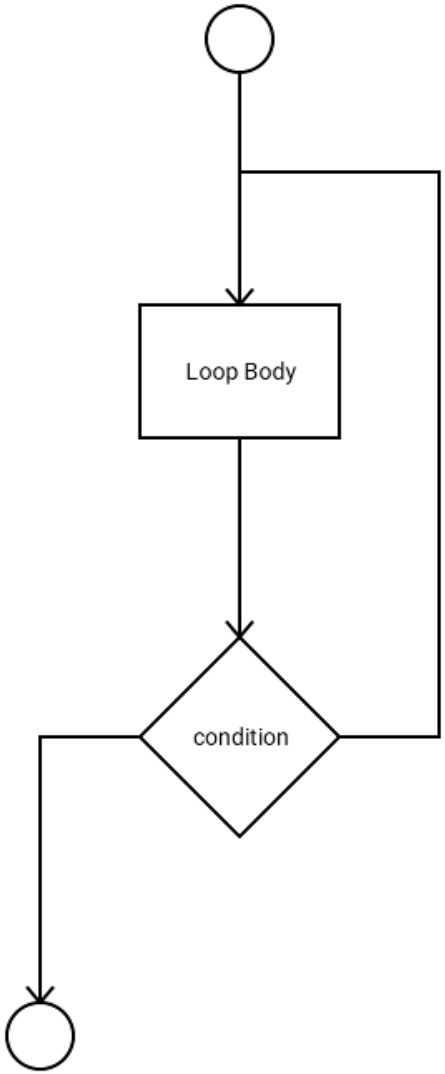


The Postcondition Loop

While Loop Syntax

```
do
        statement/block
while( condition );
```

- The `do..while` loop is called the postcondition loop.
- The condition is checked after the loop body.
- Executes 1 or more times.
- Commonly used with input validation and menus.



Example: examples/08-Loopy/validate.cpp

```
int x;        //the number to be validated.
bool valid;   //whether the choice is valid

//get a valid choice
cout << "Enter a number between 1 and 5." << endl;
do {
    //get a number
    cout << "number: ";
    cin >> x;

    //check validity
    valid = x>=1 and x<=5;

    //report errors
    if(not valid) {
        cout << "Invalid selection.  Please try again." << endl;
    }
} while(not valid);

cout << "Thank You" << endl;
```

Example: examples/08-Loopy/menu.cpp

```
int choice;

//Run the menu
do {
    cout << "1.) Option One" << endl
         << "2.) Option Two"<< endl
         << "3.) Exit" << endl
         << "Choice? ";
    cin >> choice;

    //do the selection
    if(choice == 1) {
        cout << "Option One" << endl;
    } else if(choice == 2) {
        cout << "Option Two" << endl;
    } else if(choice != 3) {
        cout << "Invalid selection, please try again." << endl;
    }
} while(choice != 3);
```

Enumerated Types

- An enumerated type or `enum` specifies a type with a fixed set of numeric values.

Enumerated Types

- An enumerated type or `enum` specifies a type with a fixed set of numeric values.
- `enum menu_choices {ONE=1, TWO, EXIT};`

Enumerated Types

- An enumerated type or `enum` specifies a type with a fixed set of numeric values.
- `enum menu_choices {ONE=1, TWO, EXIT};`
- We get three constants, `ONE`, `TWO`, `EXIT` who's values are 1, 2, and 3.

Enumerated Types

- An enumerated type or `enum` specifies a type with a fixed set of numeric values.
- `enum menu_choices {ONE=1, TWO, EXIT};`
- We get three constants, `ONE`, `TWO`, `EXIT` who's values are 1, 2, and 3.
- Often used with menus to label their selections with named constants in the code.

Example: examples/08-Loopy/menu2.cpp

```
enum menu_choice { ONE=1, TWO, EXIT };
int choice;

//Run the menu
do {
    cout << "1.) Option One" << endl
         << "2.) Option Two"<< endl
         << "3.) Exit" << endl
         << "Choice? ";
    cin >> choice;

    //do the selection
    if(choice == ONE) {
        cout << "Option One" << endl;
    } else if(choice == TWO) {
        cout << "Option Two" << endl;
    } else if(choice != EXIT) {
        cout << "Invalid selection, please try again." << endl;
    }
} while(choice != EXIT);
```

Outline

Challenge `labs/week5/guess.cpp`

We will write a guessing game!

- The computer will generate a random integer between 1 and 100.

Challenge `labs/week5/guess.cpp`

We will write a guessing game!

- The computer will generate a random integer between 1 and 100.
- The computer will ask us to guess the number.

Challenge `labs/week5/guess.cpp`

We will write a guessing game!

- The computer will generate a random integer between 1 and 100.
- The computer will ask us to guess the number.
- If we guess a number that is too low, the computer will tell us it is too low.

Challenge `labs/week5/guess.cpp`

We will write a guessing game!

- The computer will generate a random integer between 1 and 100.
- The computer will ask us to guess the number.
- If we guess a number that is too low, the computer will tell us it is too low.
- If we guess a number that is too high, the computer will let us know if it is too high.

Challenge `labs/week5/guess.cpp`

We will write a guessing game!

- The computer will generate a random integer between 1 and 100.
- The computer will ask us to guess the number.
- If we guess a number that is too low, the computer will tell us it is too low.
- If we guess a number that is too high, the computer will let us know if it is too high.
- We continue guessing until we guess the number.

Challenge `labs/week5/guess.cpp`

We will write a guessing game!

- The computer will generate a random integer between 1 and 100.
- The computer will ask us to guess the number.
- If we guess a number that is too low, the computer will tell us it is too low.
- If we guess a number that is too high, the computer will let us know if it is too high.
- We continue guessing until we guess the number.
- After we guess correctly, the computer will tell us how many tries it took us.

Challenge: `labs/week5/stock.cpp`

- Copy your most recent `stock.cpp` into your week5 directory.

Challenge: `labs/week5/stock.cpp`

- Copy your most recent `stock.cpp` into your week5 directory.
- Add an enumerated type for the menu selections.

Challenge: `labs/week5/stock.cpp`

- Copy your most recent `stock.cpp` into your week5 directory.
- Add an enumerated type for the menu selections.
- Rework your menu so it is in a loop. It should allow you to keep using the system until you choose the option that causes it to exit.

Week 5 Lab Requirements

You must have the following programs completed for full credit in week5.

1. `multiply.cpp`
2. `guess.cpp`
3. `stock.cpp` (with enum and menu loops).