02 - C++ Design and Thinking

Dr. Robert Lowe

Division of Mathematics and Computer Science
Maryville College

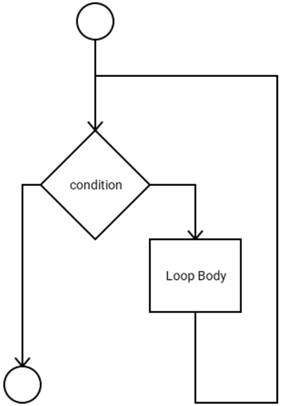
Outline

Outline

While Loop

While Loop Syntax

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

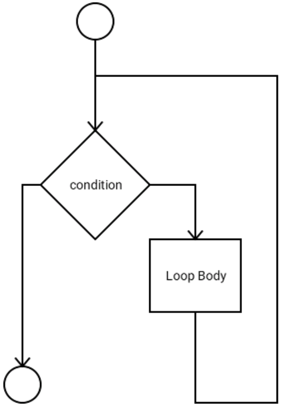


While Loop

While Loop Syntax

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

- If the *condition* is true, the loop body is executed.

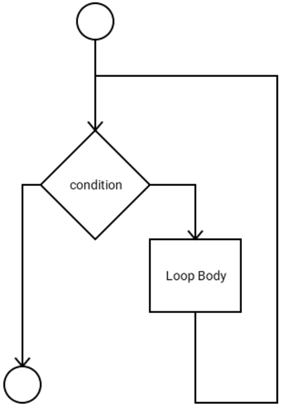


While Loop

While Loop Syntax

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

- If the *condition* is true, the loop body is executed.
- After the loop body executes, the process begins again.

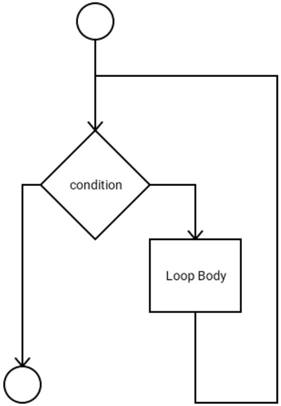


While Loop

While Loop Syntax

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

- If the *condition* is true, the loop body is executed.
- After the loop body executes, the process begins again.
- How many times will the loop body execute?

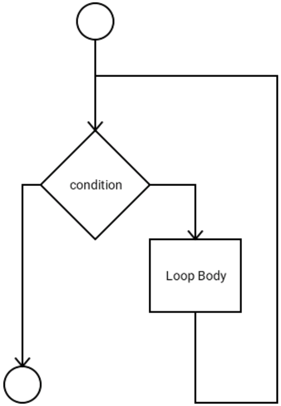


While Loop

While Loop Syntax

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

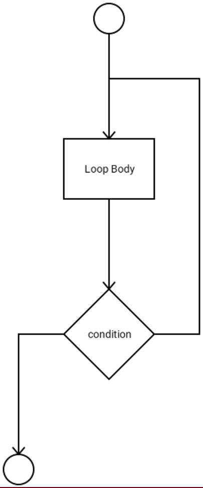
- If the *condition* is true, the loop body is executed.
- After the loop body executes, the process begins again.
- How many times will the loop body execute?
    - Zero or more times!



Do..While Loop

While Loop Syntax

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

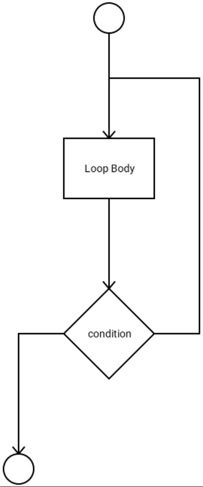


Do..While Loop

While Loop Syntax

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

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

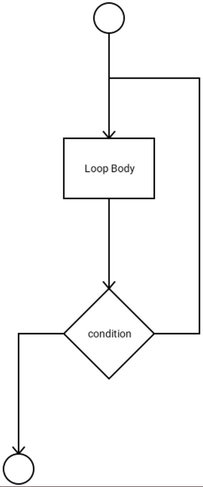


Do..While Loop

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.

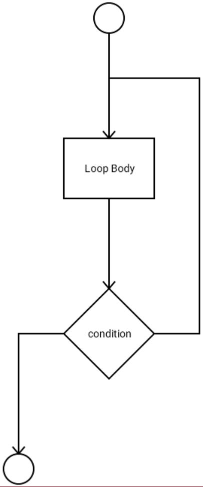


Do..While Loop

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.

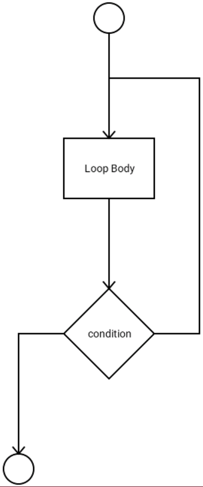


Do..While Loop

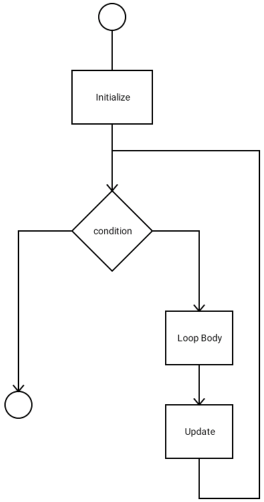
While Loop Syntax

`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



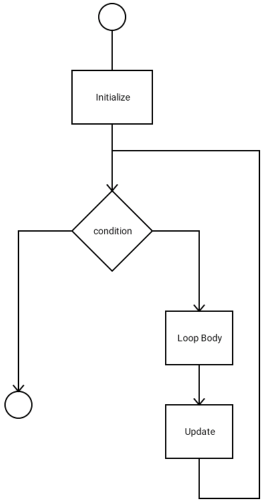
For Loop



For Loop

The For Loop

`for(` *initialize*; *condition*; *update*`)` `{`
        loop body
`}`



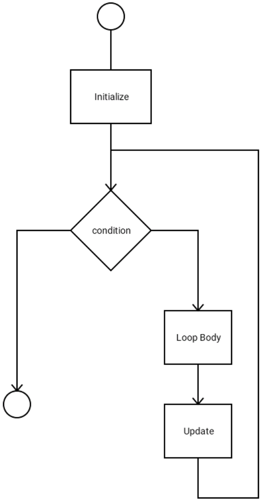
For Loop

The For Loop

`for(` *initialize*; *condition*; *update*`)  {`
        loop body
`}`

Example: Count to 10

```
for(num=0; num <= 10; num++)
{
    cout << num << endl;
}
```



Outline

The Problem

- Thus far, all known software is written by humans.

Human[1]
Temporal range: 0.35–0 Ma

Middle Pleistocene – Recent

An adult human male (left) and female (right) from the Akha tribe in Northern Thailand.

Conservation status

Least Concern (IUCN 3.1)[2]

Source: wikipedia.org

The Problem

- Thus far, all known software is written by humans.
- The human race is a member of the hominidae family.

Human[1]
Temporal range: 0.35–0 Ma

Middle Pleistocene – Recent

An adult human male (left) and female (right) from the Akha tribe in Northern Thailand.

Conservation status

Least Concern (IUCN 3.1)[2]

Source: wikipedia.org

The Problem

- Thus far, all known software is written by humans.
- The human race is a member of the hominidae family.
- We are apes.

Human[1]
Temporal range: 0.35–0 Ma

Middle Pleistocene – Recent

An adult human male (left) and female (right) from the Akha tribe in Northern Thailand.

Conservation status

Least Concern (IUCN 3.1)[2]

Source: wikipedia.org

The Problem

- Thus far, all known software is written by humans.
- The human race is a member of the hominidae family.
- We are apes.
- We are the most successful ape.

Human[1]
Temporal range: 0.35–0 Ma
Middle Pleistocene – Recent

An adult human male (left) and female (right) from the Akha tribe in Northern Thailand.

Conservation status

Least Concern (IUCN 3.1)[2]

Source: wikipedia.org

The Problem

- Thus far, all known software is written by humans.
- The human race is a member of the hominidae family.
- We are apes.
- We are the most successful ape.
- We are still apes, nonetheless.

Human[1]
Temporal range: 0.35–0 Ma
PreЄ Є O S D C P T J K PgN
Middle Pleistocene – Recent
An adult human male (left) and female (right) from the Akha tribe in Northern Thailand.
Conservation status
Extinct
Threatened
Least Concern
EX EW CR EN VU NT LC
Least Concern (IUCN 3.1)[2]

Source: wikipedia.org

The Problem

- Thus far, all known software is written by humans.
- The human race is a member of the hominidae family.
- We are apes.
- We are the most successful ape.
- We are still apes, nonetheless.
- We can hold about seven ideas in our heads at once.

Human[1]

Temporal range: 0.35–0 Ma

Middle Pleistocene – Recent

An adult human male (left) and female (right) from the Akha tribe in Northern Thailand.

Conservation status

Least Concern (IUCN 3.1)[2]

Source: wikipedia.org

The Problem

- Thus far, all known software is written by humans.
- The human race is a member of the hominidae family.
- We are apes.
- We are the most successful ape.
- We are still apes, nonetheless.
- We can hold about seven ideas in our heads at once.
- This is insufficient for almost all useful programming tasks.

Human[1]
Temporal range: 0.35–0 Ma

Middle Pleistocene – Recent

An adult human male (left) and female (right) from the Akha tribe in Northern Thailand.

Conservation status

Least Concern (IUCN 3.1)[2]

Source: wikipedia.org

Function Definition

Function Syntax

```
return_type name( parameters )
{
    //function body
}
```

Function Definition

Function Syntax

```
return_type name( parameters )
{
    //function body
}
```

Function Definition

Function Syntax

```
return_type name( parameters )
{
    //function body
}
```

- A function is a block of code that can be called multiple times.

Function Definition

Function Syntax

```
return_type name( parameters )
{
    //function body
}
```

- A function is a block of code that can be called multiple times.
- A function's signature consists of the following:

Function Definition

Function Syntax

```
return_type name( parameters )
{
    //function body
}
```

- A function is a block of code that can be called multiple times.
- A function's signature consists of the following:

return type	This is the type of value the function evaluates to when it is used in an expression.

Function Definition

Function Syntax

```
return_type name( parameters )
{
    //function body
}
```

- A function is a block of code that can be called multiple times.
- A function's signature consists of the following:

return type	This is the type of value the function evaluates to when it is used in an expression.
name	The identifier which names the function.

Function Definition

Function Syntax

```
return_type name( parameters )
{
    //function body
}
```

- A function is a block of code that can be called multiple times.
- A function's signature consists of the following:

return type	This is the type of value the function evaluates to when it is used in an expression.
name	The identifier which names the function.
parameters	The local variables which receive the arguments of the function.

Function Prototypes

- Function prototypes allow you to declare a function before it is defined.

Function Prototypes

- Function prototypes allow you to declare a function before it is defined.
- This is a sort of “contract” between you and the compiler.

Function Prototypes

- Function prototypes allow you to declare a function before it is defined.
- This is a sort of “contract” between you and the compiler.
- This allows you to have functions in any order in the file.

Function Prototypes

- Function prototypes allow you to declare a function before it is defined.
- This is a sort of “contract” between you and the compiler.
- This allows you to have functions in any order in the file.
- Change the first few lines of `roman.cpp` so it reads as follows:

```
#include <iostream>

using namespace std;

//function prototypes
void print_roman_numeral(int value);
```

Gluing it Together With Header Files

- A function must be declared before it can be used.

Gluing it Together With Header Files

- A function must be declared before it can be used.
- Because the definitions are in a separate file, they are not declared in the file that contains our main function.

Gluing it Together With Header Files

- A function must be declared before it can be used.
- Because the definitions are in a separate file, they are not declared in the file that contains our main function.
- We can solve this with prototypes.

Gluing it Together With Header Files

- A function must be declared before it can be used.
- Because the definitions are in a separate file, they are not declared in the file that contains our main function.
- We can solve this with prototypes.
- Repeating prototypes in every file is painful.

Gluing it Together With Header Files

- A function must be declared before it can be used.
- Because the definitions are in a separate file, they are not declared in the file that contains our main function.
- We can solve this with prototypes.
- Repeating prototypes in every file is painful.
- Enter the header file!

Gluing it Together With Header Files

- A function must be declared before it can be used.
- Because the definitions are in a separate file, they are not declared in the file that contains our main function.
- We can solve this with prototypes.
- Repeating prototypes in every file is painful.
- Enter the header file!
- A header file usually has a `.h` extension and contains:

Gluing it Together With Header Files

- A function must be declared before it can be used.
- Because the definitions are in a separate file, they are not declared in the file that contains our main function.
- We can solve this with prototypes.
- Repeating prototypes in every file is painful.
- Enter the header file!
- A header file usually has a `.h` extension and contains:
    - Function Prototypes

Gluing it Together With Header Files

- A function must be declared before it can be used.
- Because the definitions are in a separate file, they are not declared in the file that contains our main function.
- We can solve this with prototypes.
- Repeating prototypes in every file is painful.
- Enter the header file!
- A header file usually has a `.h` extension and contains:
  - Function Prototypes
  - Constants

Gluing it Together With Header Files

- A function must be declared before it can be used.
- Because the definitions are in a separate file, they are not declared in the file that contains our main function.
- We can solve this with prototypes.
- Repeating prototypes in every file is painful.
- Enter the header file!
- A header file usually has a `.h` extension and contains:
  - Function Prototypes
  - Constants
  - Type Definitions

Outline

Makefile – Explicit Recipes

```
sodasim: sodasim.o soda-machine.o
    g++ -o sodasim sodasim.o soda-machine.o

sodasim.o: sodasim.cpp soda-machine.h
soda-machine.o: soda-machine.cpp soda-machine.h
```

Some Predefined Variables

- The make syntax is itself a scripting language.

Some Predefined Variables

- The make syntax is itself a scripting language.
- Variables begin with dollar signs $.

Some Predefined Variables

- The make syntax is itself a scripting language.
- Variables begin with dollar signs $.
- There are several pre-defined variables, the two most commonly used ones are:

Some Predefined Variables

- The make syntax is itself a scripting language.
- Variables begin with dollar signs $.
- There are several pre-defined variables, the two most commonly used ones are:
    - `$@` – The name of the target

Some Predefined Variables

- The make syntax is itself a scripting language.
- Variables begin with dollar signs $.
- There are several pre-defined variables, the two most commonly used ones are:
  - `$@` – The name of the target
  - `$^` – The list of all ingredients

Some Predefined Variables

- The make syntax is itself a scripting language.
- Variables begin with dollar signs $.
- There are several pre-defined variables, the two most commonly used ones are:
  - `$@` – The name of the target
  - `$^` – The list of all ingredients
- We could simplify the sodasim `Makefile` like so:

```
sodasim: sodasim.o soda-machine.o
    g++ -o $@ $^

sodasim.o: sodasim.cpp soda-machine.h
soda-machine.o: soda-machine.cpp soda-machine.h
```

Example Makefile – Address Book

```
TARGETS=stock

#application builds
all: $(TARGETS)
stock: iofun.o main.o stock.o transaction.o portfolio.o
        g++ -o $@ $^

#object files
iofun.o: iofun.h iofun.cpp
main.o: main.cpp iofun.h stock.h transaction.h portfolio.h
stock.o: stock.h stock.cpp
transction.o: transaction.cpp transaction.h
portfolio.o: portfolio.cpp portfolio.h


#delete all binaries
clean:
        rm -f *.o $(TARGETS)
```

Outline

Programming Project 5.9

Programming Project 5.9 from Big C++

Write a program that, given a month and year, prints a calendar, such as

```
      June 2016
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

Make a helper function to print the header and a helper function to print each row.