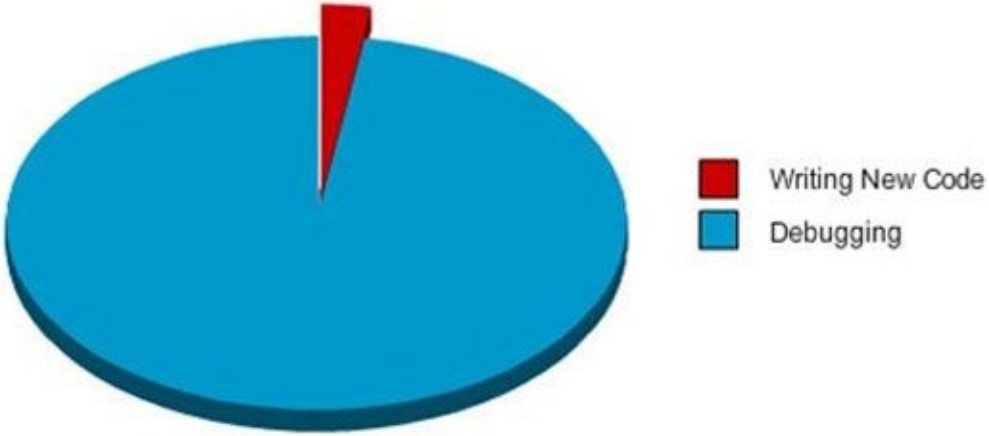
Programming



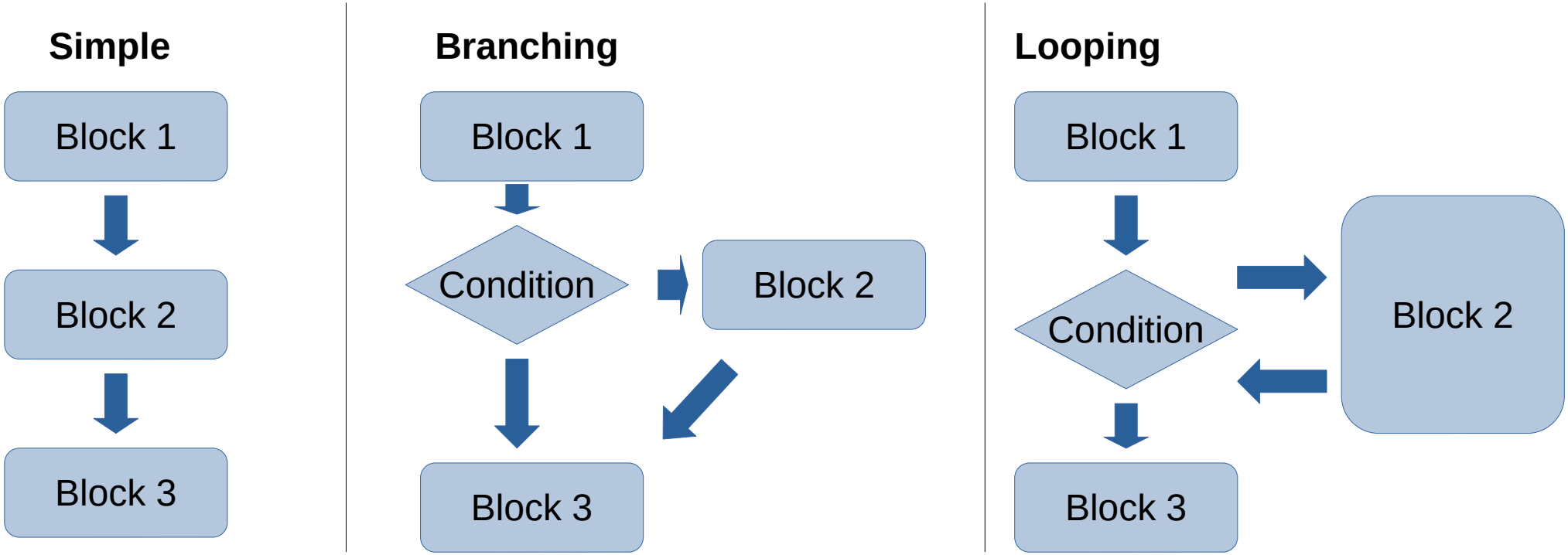
CS1428
Foundation of Computer Science

Lecture 5: Looping

What is Looping?

- Our simplest programs executed each instructions one after the other.
- **Branching** gave us a tool to conditionally execute some statements and not others.
- **Looping** is a tool which let’s us conditionally repeat some statements in our programs.

Looping Flowchart



While Loop Syntax

- The C++ keyword **while** is one common way to implement looping in a program.
- Every **while** must be followed by a condition in (parenthesis).
- Without {curly brackets}, then only the next statement is in the body of the loop. With {curly brackets}, everything in the brackets is in the loop.
- The variable that the condition depends on must be changed in the body of the loop to avoid an **infinite loop**.

While Loop Body

```
while(a>1)
    cout << "Only this statment is in the loop body";
    cout << "This one is not";

while(a<10){
    cout << "This statement is in the loop body";
    cout << "So is this one."
}
```

Always Update the Loop Variable

```
int a=3;
while(a>0){
    cout << "a=" << a << endl;
    a--;
}
```

Loop Output

```
/home/gentry/Desktop/1428_testDir/junk
a=3
a=2
a=1

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Infinite Loops

```
int a=3;
while(a>0){
    cout << "a=" << a << endl;
    a++;
}
```

Infinite Loop Output

This is where I manually stopped the program.

```
/home/gentry/Desktop/1428_testDir/junk
a=182086
a=182087
a=182088
a=182089
a=182090
a=182091
a=182092
a=182093
a=182094
a=182095
a=182096
a=182097
a=182098
a=182099
a=182100
a=182101
a=182102
a=182103
a=182104
a=182105
a=182106
a=182107
a=182108
a=182109
a=182110
a=182111
a=182112^C
Process returned -1 (0xFFFFFFFF)   execution time : 1.784 s
Press ENTER to continue.
```

Do...While Loop Syntax

- The C++ keyword **do...while** also implements looping in programs but checks the condition **after** the loop body.
- {curly brackets} can be used to enclose multiple statements, otherwise only one statement is in the loop body.
- The loop body is written between the **do** and the **while**.
- The condition is written following the while, in (parenthesis).
- The condition must be terminated with a semicolon;

Do...While Loop Body

```
do cout << "Only this line is in the loop";
while(a < 5); //remember this semicolon

do{
    cout << "This line is in the loop";
    cout << "So is this one".
}while(a<5); //remember this semicolon
```

Do...While Runs at Least Once

```
do{
    cout << "You'll see this line." << endl;
}while(false);

while(false){
    cout << "But not this one." << endl;
}
```

Which Loop had Visible Output?

```
/home/gentry/Desktop/1428_testDir/junk
You'll see this line.

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Boolean Variables can be Conditions

```
bool loop_var = true;
while(loop_var){
    cout << "Loop is running" << endl;
    loop_var = !loop_var;
}
```

Loop Output

```
/home/gentry/Desktop/1428_testDir/junk

Loop is running

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

For Loop Syntax

- The C++ keyword **for** loops over a **fixed range** of values.
- {curly brackets} can be used to enclose multiple statements, otherwise only one statement is in the loop body.
- The for loop lets us create a loop variable, define a condition, and define the update to the variable all at the same time.
- **for(declaration; condition; update){loop body}**

Counting with a For Loop

```
int main(){

    for(int i = 1; i <= 5; i++){
        cout << "Run number: " << i << endl;
    }

}
```

Counting Output

```
/home/gentry/Desktop/1428_testDir/junk

Run number: 1
Run number: 2
Run number: 3
Run number: 4
Run number: 5

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Unrolling a For Loop

```
for(int i = 1; i <= 5; i++)
```

- **Start** → create a variable named i and give it the value 1
- i is less than or equal to 5 so do the loop body
- After the body has run, update i to 2.
- 2 is less that or equal to 5, run again, update i to 3.
- 3 is less that or equal to 5, run again, update i to 4.
- 4 is less that or equal to 5, run again, update i to 5.
- 5 is less that or equal to 5, run again, update i to 6.
- 6 is not less than or equal to 5, break the loop.

Converting For to While

```
cout << "---For Loop---" << endl;
for(int i = 1; i <= 5; i++){
    cout << "Run number: " << i << endl;
}

cout << endl << "---While Loop---" << endl;
int i = 1;
while(i <= 5){
    cout << "Run number: " << i << endl;
    i++;
}
```

Same Output

```
/home/gentry/Desktop/1428_testDir/junk

---For Loop---
Run number: 1
Run number: 2
Run number: 3
Run number: 4
Run number: 5

---While Loop---
Run number: 1
Run number: 2
Run number: 3
Run number: 4
Run number: 5

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Converting While to Do...While

```
cout << "---While Loop---" << endl;
int i = 1;
while(i <= 5){
    cout << "Run number: " << i << endl;
    i++;
}

cout << endl << "---Do...While Loop---" << endl;
i = 1;
do{
    cout << "Run number: " << i << endl;
    i++;
}while(i<=5);
```

Same Output

```
/home/gentry/Desktop/1428_testDir/junk
---While Loop---
Run number: 1
Run number: 2
Run number: 3
Run number: 4
Run number: 5

---Do...While Loop---
Run number: 1
Run number: 2
Run number: 3
Run number: 4
Run number: 5

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Which Loop to Use

- Every loop can **usually** be re-written as one of the other kinds of loop, so choosing which loop to use often comes down to programmer preference.
- **while** is a good choice when the body may not run at all, or for long loop bodies to make the condition easily visible.
- **do...while** is a good choice when the loop variable is initialized inside of the loop body.
- **for** is a good choice when iterating over a fixed range of numbers.

While is a Good Choice

```
ifstream in_file("Sample.txt");
string from_file;
//This loop will not run if the file is empty.
while(!in_file.eof()){
    in_file >> from_file;
}
```

Do...While is a Good Choice

```
int player_choice, secret_number=13;
//We can't check the condition until the body runs
do{
    cout << "Please guess a number: ";
    cin >> player_choice;
}while(player_choice != secret_number);
cout << "You have guessed the correct number.";
```

For is a Good Choice

```
ofstream flavor_list("Ice_cream.txt");
cout << "List your 10 favorite Ice Cream flavors: ";
string flavor;
//This loop always runs 10 times
for(int i = 0; i < 10; i++){
    cin >> flavor;
    flavor_list << flavor;
}
```

Nesting Loops

- Just like branching, loops can be “nested” inside of one another.
- The inner loop will run completely for every one iteration of the outer loop.
- Nested loops can add a lot of run time to programs.

Nested Loops

```
int main(){
    int x;
    for(int i = 0; i<100000; i++){
        for(int j = 0; j<100000; j++){
            x = rand();
        }
    }
    cout << "After 10,000,000,000 assignments x is " << x << endl;
    return 0;
}
```

Nested Output

```
/home/gentry/Desktop/1428_testDir/junk
After 10,000,000,000 assignments x is 2045180722

Process returned 0 (0x0)   execution time : 71.449 s
Press ENTER to continue.
```

I waited more than a minute for this to run.

Tracking Iterations

```
int main(){
    int outter_counter = 0;
    int inner_counter;
    while (outter_counter < 3){
        inner_counter = 0;
        cout << "### Outer Count = " << outter_counter << " ###" << endl;
        while(inner_counter < 3){
            cout << "Inner Count " << inner_counter << endl;
            inner_counter++;
        }
        outter_counter++;
    }
}
```

Nested Output

```
/home/gentry/Desktop/1428_testDir/junk
### Outer Count = 0 ###
Inner Count 0
Inner Count 1
Inner Count 2
### Outer Count = 1 ###
Inner Count 0
Inner Count 1
Inner Count 2
### Outer Count = 2 ###
Inner Count 0
Inner Count 1
Inner Count 2

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Who Noticed This?

Two variables with the same name?

```
cout << "---For Loop---" << endl;
for(int i = 1; i <= 5; i++){
    cout << "Run number: " << i << endl;
}

cout << endl << "---While Loop---" << endl;
int i = 1;
while(i <= 5){
    cout << "Run number: " << i << endl;
    i++;
}
```

Scope

- In C++, many variables can have the same name. Each new variable “shadows” the other variables that share its name.
- The **scope** refers to the portion of the program where a variable can be referenced.
- A variable name must be unique within a particular scope.

3 Layers of Scope

```
int x = 1;

int main(){
    cout << "x at global scope " << x << endl;
    int x = 2;
    cout << "x at function scope " << x << endl;
    if(true){
        int x = 3;
        cout << "x at block scope " << x << endl;
    }
    return 0;
}
```

Same Output

```
/home/gentry/Desktop/1428_testDir/junk
x at global scope 1
x at function scope 2
x at block scope 3

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

3 Layers of Scope

- **Global**- visible to the entire program.
- **Function**- only visible to one function (like **main**)
- **Block**- can only be referenced inside of the block (e.g. an **if...else** statement, or a loop) that its declared in.
- Block and functions are both types of **local scope.**

Global Scope

- Generally not a good idea! (in C++)
- Global variables can accidentally be referenced in any part of the program.
- Global variables can expose “private” information in accessible regions of memory.
- Global <u>constants</u> are perfectly acceptable and encouraged.

Global Constants

```
const int CLASS_SIZE = 20;
const string FILENAME = "CS1428/roster.txt";

int main(){
    ofstream roster_file(FILENAME);
    string name;
    cout << "Enter a name for " << CLASS_SIZE << " students" << endl;
    for (int i = 0; i < CLASS_SIZE; i++){
        cout << "Student #" << i << ": ";
        cin >> name;
        roster_file << name;
    }
    cout << CLASS_SIZE << " student names recorded." << endl;
    return 0;
}
```

Referencing Variables at Higher Scope

```
int x = 1;

int main(){
    int x = 2;
    cout << "Local scope x: " << x << endl;
    cout << "Global scope x: " << ::x << endl;
    return 0;
}
```

Global Scope Output

```
/home/gentry/Desktop/1428_testDir/junk

Local scope x: 2
Global scope x: 1

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.
```

Style Guide, Comments

- Our programs are starting to get more complicated, making good comments important to include.
- One comment should describe the action being performed by several statements or a whole block.

```
int user_input;
//take user input until 'q' or 'Q' is entered
do{
    cin >> user_input;
}while(user_input != 'q' && user_input != 'Q');
```