Unit 1b – Processing Information using Expressions

Mark Redekopp

VARIABLES AND ASSIGNMENT

Variable Review: I Do Declare

- Unlike some other languages (e.g. Python) you must do a **one-time declaration** of a variable before using it
  - Like renting an apartment or storage unit
- C++ is a **strongly-typed** language which means…
  - You *cannot change* what type of value the variable stores); this is because in C++ a variable name corresponds to a reserved, fixed-size memory location that only fits that specific type

```
#include <iostream>
using namespace std;
int main() {
  v = 2;      // ERROR: x assigned before
              // it is declared
  int y = 2; // Must declare with type first

  y = "pi is"; // Error: y declared as int
               // cannot be assigned a string
  y = 3;      // Change value stored in y

  cout << y << endl;
  return 0;
}
```

C++ is "strongly-typed" and requires variables to be declared before being used.

```
def main():
    y = 2       # x stores an integer
    y = "pi is" # x changes to store a string
    print(y)
```

Python does not require explicitly declaring and typing a variable

C++ Types, Storage, and Range

C Type	Usage	Byte(s)/Bits	Range

`int`
`unsigned int`
	Integer values	4 / 32	-2 billion to +2 billion
			0 to +4 billion

`char`	Text character or (small integral value)	1 / 8	ASCII characters -128 to +127
~~`float`~~ `double`	Rational/real values	~~4 / 32~~ 8 / 64	~~7 significant digits *~~ ~~$10^{+/-308}$~~ 16 significant digits * $10^{+/-308}$

`string`
`char[]`
	Arbitrary text	Arbitrary
		1 byte per char
			-

`bool`	True/False value	1 / 8	true / false

```
#include <string>
using namespace std;

int main()
{
  int          a = -1;
  unsigned int b = 2;
  char c = 'A';  // 'A'=65

  float  d1 = 1.5;
  double d2 = 3.14;

  char e[6] = "Hello";
  string f  = "Goodbye";

  bool g = true;

  // ...
}
```

Variable

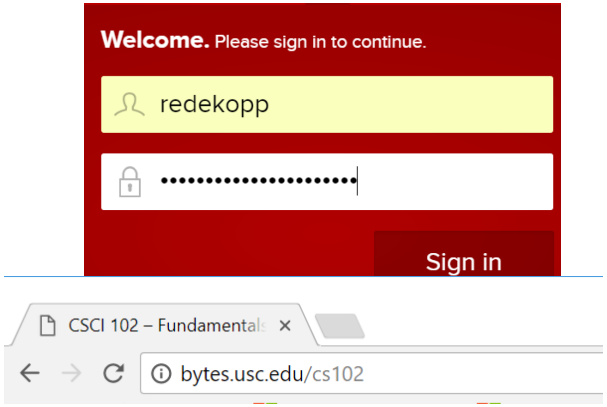
Constant

Constant

When To Introduce a Variable

- When a value will be input (via cin) and/or change at run-time (as the program executes)

```
______ username, password;
cin >> username >> password;
```

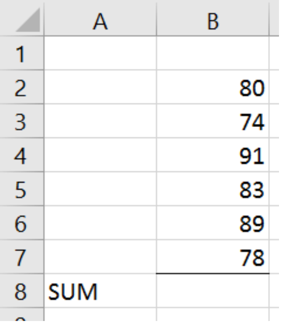


Welcome. Please sign in to continue.

redekopp

••••••••••••••••••••••

- When a value is computed/updated at one time and used (many times) later



	A	B
1		
2		80
3		74
4		91
5		83
6		89
7		78
8	SUM	

- To make the code more readable by another human

```
double a = (x+34) * (n*6.25);

// readability of above vs. below

double height = x + 34;
double width =  n * 6.25;
double area = height * width;
```

What Variables Might Be Needed

- Video playback (YouTube player)
  -

- Calculator App
  -

C/C++ Variables

- Variables have a:
  - type [`int, char, unsigned int, float, double`, etc.]
  - name/identifier that the **programmer will use** to reference the value in that memory location [e.g. `x, myVariable, num_dozens`, etc.]
    - Identifiers must start with [A-Z, a-z, or an underscore '_'] and can then contain any alphanumeric character [0-9, A-Z, a-z, _] (but no punctuation other than underscores)
    - Use descriptive names (e.g. numStudents, doneFlag)
    - Avoid cryptic names (myvar1, a_thing)
  - location [the address in memory where it is allocated which the **computer will use** to access the value]
  - Value
- Reminder: You must declare a variable before using it

What's in a name?

To give descriptive names we often need to use more than 1 word/term. But we can't use spaces in our identifier names. Thus, most programmers use either camel-case or snake-case to write compound names

Camel case: Capitalize the first letter of each word (with the possible exception of the first word)

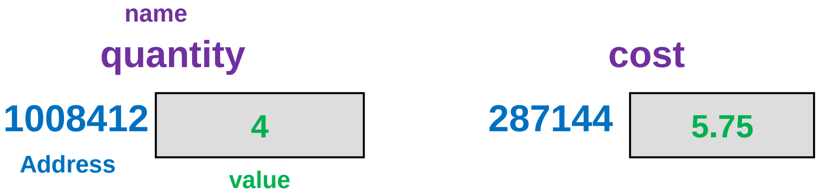
myVariable, isHighEnough

Snake case: Separate each word with an underscore '_'

my_variable, is_high_enough

Code

```
int quantity = 4;
double cost = 5.75;
cout << quantity*cost << endl;
```



VARIABLE ASSIGNMENT USING '=' OPERATOR

Assignment operator (=)

- Assignment operator ('=') updates what is stored in a variable's memory (storage location)
- Key to understanding assignment:
  - tfel ot thgir krow

```
int x = 1;
x = x + 3;
```

Assignment operator (=)

- Syntax:

```
variable = expression;
```

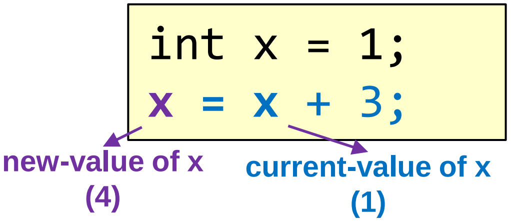
(LHS) (RHS)

  - LHS = Left Hand-Side, RHS = Right Hand Side

- Should be read: Store the value of *<expression>* into memory location of *<variable>*

  - `z = x + y - (2*z);`
  - If variable appears on both sides, we use the old/current value of the variable on the RHS

- = does **NOT** mean "compare for equality"; that is the == operator



```
int x = 1;
x = x + 3;
```

new-value of x (4)

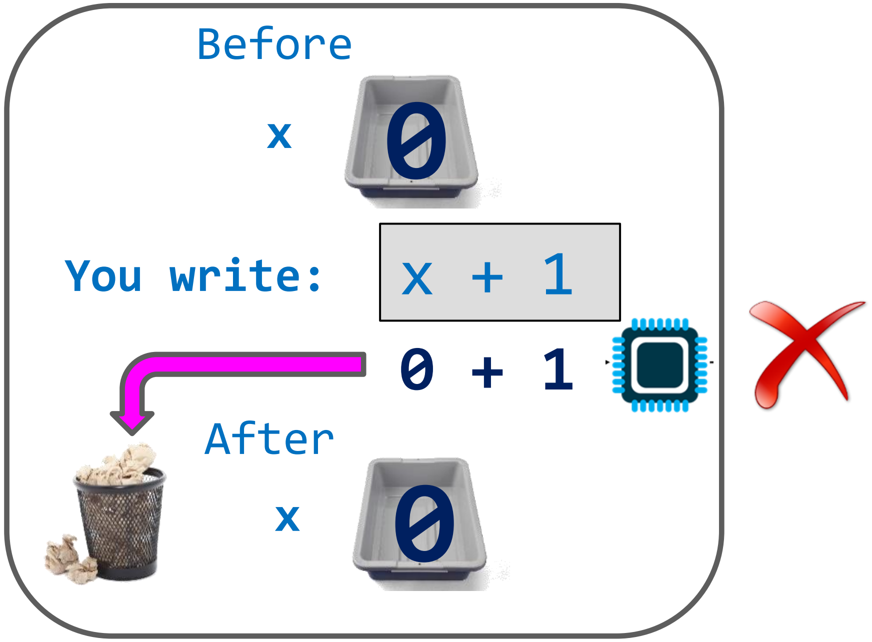
current-value of x (1)

Evaluate **everything** on the right-hand side (RHS) before considering the left-hand side (LHS)

Common Mistake: Forgetting to Assign

- Without assignment values are computed and then forgotten ⚠

```
– x + 1;       // Takes x's value and adds 1 but DOES NOT
               //  update x (just throws the result away)
– x = x + 1;   // Using assignment, x actually updates
```



Before

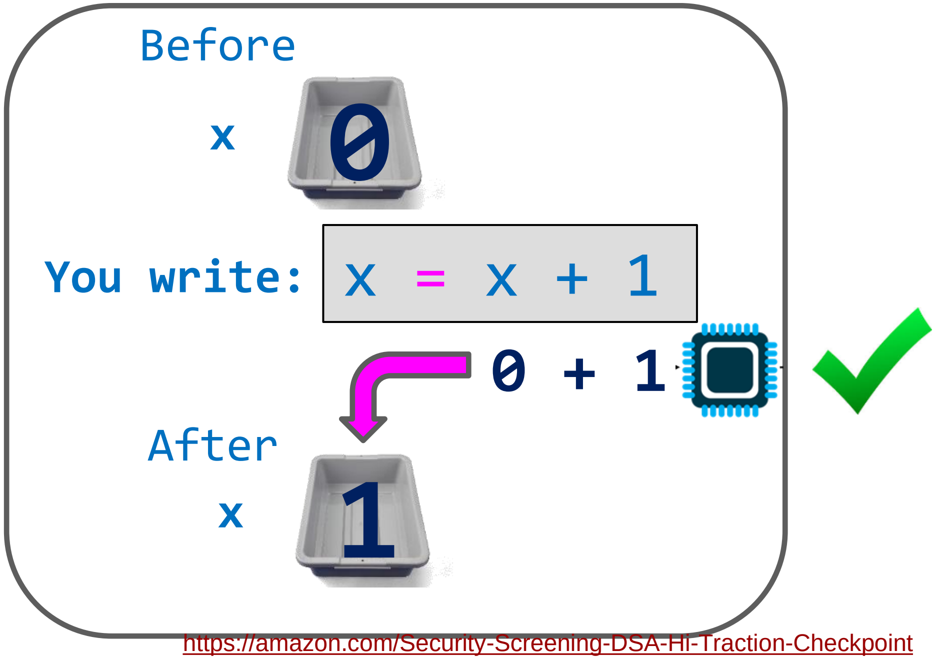
x 0

You write: x + 1

0 + 1

After

x 0



Before

x 0

You write: x = x + 1

0 + 1

After

x 1

https://amazon.com/Security-Screening-DSA-Hi-Traction-Checkpoint

This Photo by Unknown Author is licensed under CC BY

Common Mistake: Forgetting to Initialize

- Declaring a variable **DOES NOT initialize its value to 0** or some other known value.
- In fact, an uninitialized variable will contain **random data/garbage**.
- It is at least good practice, if not necessary, to initialize your variables
  - **Exception**: If you are just going to perform a `cin` command to that variable it is probably fine to leave it uninitialized (but you are welcome to set it to 0 or other value).

```
#include <iostream>
using namespace std;
int main() {
  int x;      // BAD: x has random garbage
              //   value
  x = x + 3; // What will x be after adding 3?

  int y = 2; // GOOD: declare and init.
              //   together
  y = y + 3; // What will y be after adding 3?

  int z;      // OK: z is random garbage...
  cin >> z;  //   ...but cin will init z

  return 0;
}
```

C++ is "strongly-typed" and requires variables to be declared before being used.

`int x;`

104	01101000
105	11010001
106	01101000
107	11010001

Assignment (=) Operator Summary

- We can use `=` to update a variable as often as we like

```
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
  int w=0;  // variables don't have to
  char x='z'; //  be initialized when declared

  w = 300;
  x = 'a';
  cout << w << " " << x << endl;

  w = -75;
  x = '!';
  cout << w << " " << x << endl;
  return 0;
}
```

Output:

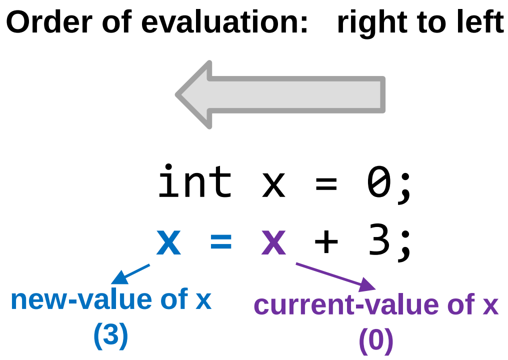
```
300 a
-75 !
```

Assignment is one of the most common operations in programs

Exercise: Trace the Code Below

- Variables can be used in expressions and be operands for arithmetic and logic
- See inset below on how to interpret a variable's usage based on which side of the assignment operator it is used

```
// iostream allows access to 'cout'
#include <iostream>
using namespace std;

// Execution always starts at the main() function
int main()
{
  int dozens = 3;
  double gpa = 2.0;

  int num = 12 * dozens;
  gpa = (2 * 4.0) + (4 * 3.7);  // gpa updated to 22.8
  gpa = gpa / 6;  // integer or double division?

  cout << dozens << " dozen is " << num << " items." << endl;
  cout << "Your gpa is " << gpa << endl;
  return 0;
}
```



Order of evaluation: right to left

```
int x = 0;
x = x + 3;
```

new-value of x (3)

current-value of x (0)

Semantics of variable usage:

- **Right-side of assignment: Substitute/use the current value stored in the variable**
- **Left-side of assignment: variable is the destination location where the result of the right side will be stored**

More Exercises

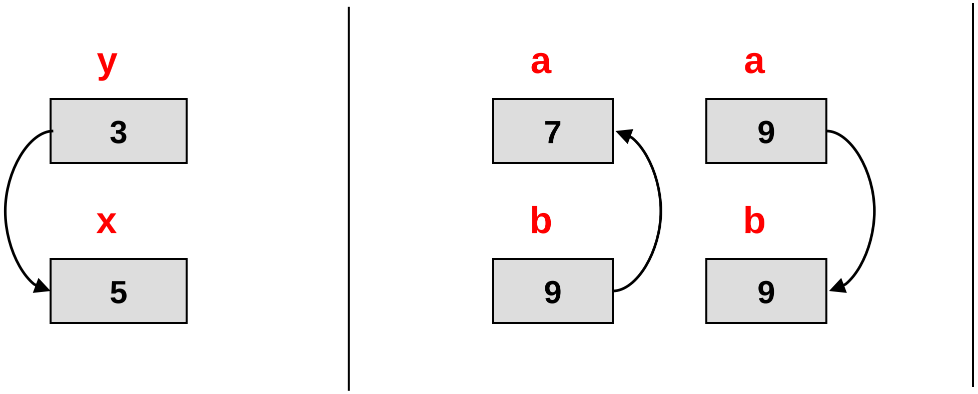
- What is printed by the following two programs?

```
#include <iostream>
using namespace std;

int main()
{
  int value = 1;
  value = (value + 5) * (value – 3);
  cout << value << endl;

  double amount = 2.5;
  value = 7;
  amount = value + 6 / amount;
  cout << amount << endl;

  cout << value % 3 << endl;
  return 0;
}
```

```
#include <iostream>
using namespace std;

int main()
{
  int x = 5;
  int y = 3;
  double z = x % y * 6 + x / y;

  cout << z << endl;

  z = 1.0 / 4 * (z – x) + y;
  cout << z << endl;

  return 0;
}
```

Important: Assignment Means Copy

- Assigning a variable makes a **copy**
  - It leaves the source variable unchanged
  - Is performed immediately and takes effect before the next statement
- Order/sequence MATTERS!
  - 1 assignment statement affects subsequent expressions
- Challenge: Swap the value of 2 variables

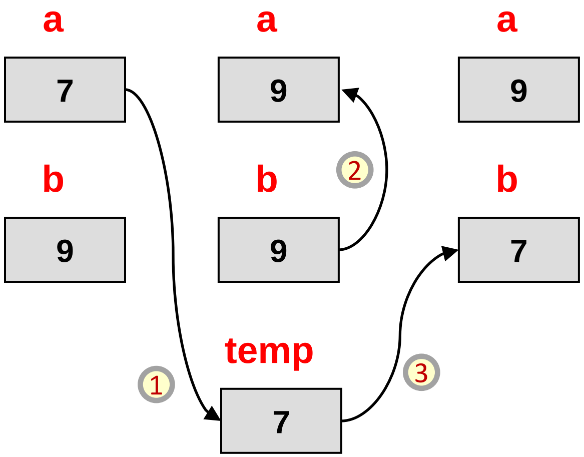


```
int main()
{
  int x = 5, y = 3;
  x = y;    // copy y into x
            // y still has 3
  return 0;
}
```

```
int main()
{
  int a = 7, b = 9;

  // now consider swapping
  // the value of 2 variables
  a = b;
  b = a;

  return 0;
}
```

More Assignments

- Assigning a variable makes a **copy**
  - It leaves the source variable unchanged
- Example: Swap the value of 2 variables
  - Easiest method: Use a 3rd temporary variable to save one value and then replace that variable
- Challenge: 4swap exercise



```
int main()
{
 int a = 7, b = 9, temp;

 // let's try again
 temp = a;
 a = b;
 b = temp;

 cout << a << " " << b << endl;
 return 0;
}
```

Shortcut Assignment Statements

- A common task is to update a variable by adding, subtracting, multiplying, etc. some value to it
  - `x = x + 4;`
  - `y = y * 2.5;`
- C/C++ provide a shortcut for writing these statements:
  - `x += 4;`
  - `y *= 2.5;`
- The substitution is:
  - `var op= expr;`
  - Becomes `var = var op expr;`
- **Shorthand operators** exist for most operators:

  `+=, -=, *=, /=, %=, &=, …`

```
#include <iostream>
using namespace std;

int main()
{
  int x = 1;
  double y = 3.75;

  x += 5;    // x updates to 6
  y -= 2.25; // y updates to 1.5
  x /= 3;    // x updates to 2
  y *= 2.0   // y updates to 3.0

  return 0;
}
```

Post-Increment/Decrement

- Adding 1 to a variable (e.g. `x += 1`) and subtracting 1 from a variable (e.g. `x -= 1`) are extremely common operations (especially when we cover loops).
- The `++` and `--` operators offer a shortcut to "increment-by-1" or "decrement-by-1"
  - Performs (`x += 1`) or (`x -= 1`)
  - `x++; // If x was 2 it will be updated to 3 (x = x + 1)`
  - `x--; // If x was 2 it will be updated to 1 (x = x – 1)`

- Note: There are some nuances to this operator and an alternative known as **PRE**-increment/decrement that we will discuss in future lectures, but this is sufficient for now.

CASTING AND USING MATH LIBRARY FUNCTIONS

Casting Motivation

- ***Def***. casting: *Temporarily converting the type of a data value*
- What is the result of 5 + 3/2 ?
  - To achieve the correct answer for 5 + 3 / 2 we could…
- Use **implicit** casting (mixed expression)
  - Could just write 5 + 3.0 / 2
    - If an operator is applied to mixed type inputs, less expressive type is automatically and implicitly cast (promoted) to the more expressive (int is promoted to double)
- But what if instead of constants we have variables

```
– int x=5, y=3, z=2;
  x + y/z;   // Won't work & you can't write y.0
```

- We can perform an **explicit** cast using either the C or C++ syntax

```
– x + (double) y  /  z;              // C   style casting method
– x + static_cast<double>(y)  /  z ; // C++ style casting method
```

- BE CAREFUL!! This won't yield the 6.5 answer you expect.

```
– x + static_cast<double>(y/z);  // Why not?
```

Math & Other Library Functions

- C++ predefines a variety of functions for you. Here are a few of them:
  - sqrt(x): returns the square root of x (in <cmath>)
  - pow(x, y): returns x^y, or x to the power y (in <cmath>)
  - sin(x)/cos(x)/tan(s): returns the sine of x if x is in radians (in <cmath>)
  - abs(x): returns the absolute value of x (in <cstdlib>)
  - max(x, y) and min(x,y): returns the maximum/minimum of x and y (in <algorithm>)
- You call these by writing them similarly to how you would use a function in mathematics [using parentheses for the inputs (aka) arguments]
- Result is replaced into bigger expression
- Must #include the correct library
  - #includes tell the compiler about the various pre-defined functions that your program may choose to call

```
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;

int main()
{
  // can call functions
  //  in an assignment
  double res = cos(0); // res = 1.0

  // can call functions in an
  //  expression
  res =  sqrt(2) / 2; // res = 1.414/2

  cout << max(34, 56) << endl;
  // outputs 56

  return 0;
}
```

http://www.cplusplus.com/reference/cmath/

#include Directive

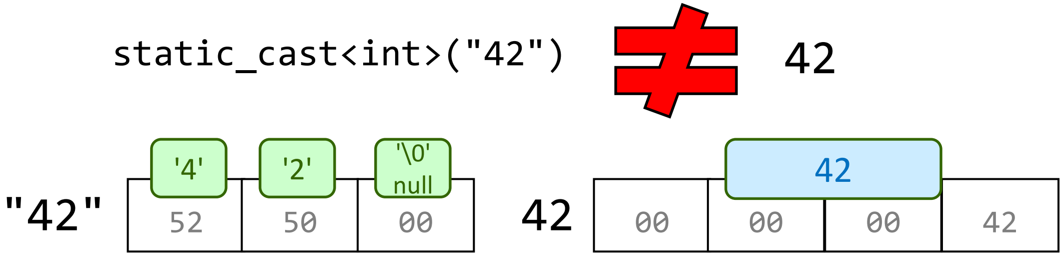
- Common usage: To include “header files” that allow us to access functions defined in a separate file or library
- For pure C compilers, we include a C header file with its filename: #include <stdlib.h>
- For C++ compilers, we include a C header file without the .h extension and prepend a ‘c’: #include <cstdlib>

C	Description	C++	Description
stdio.h **cstdio**	C Input/Output/File access (printf, fopen, snprintf, etc.)	**iostream**	I/O and File streams (cin, cout, cerr)
stdlib.h **cstdlib**	rand(), Memory allocation, etc.	**algorithm**	Common data processing tasks/algorithms (find, sort, min/max)
string.h **cstring**	C-string library functions that operate on character arrays	**string**	C++ string class that defines the ‘string’ object
math.h **cmath**	Math functions: sin(), pow(), etc.	**vector**	Array-like container class

Common Casting Errors

- Only changes the type **temporarily** for the sake of the expression (not a permanent type change)
- Casting only really works on numeric types and NOT strings
  - Different than many other languages like Python
  - When converting to/from a string, do NOT use casting, but functions from the string library (to_string(), stoi(), stod(), etc.)

```
#include <iostream>
#include <string>
using namespace std;
int main() {

 double a = 3.6;
 int b = static_cast<int>(a) / 2;
    // Works! b = 1 (casts 3.6 to 3)
    // but a is still a double: 3.6
 int c = 123;
 string d = static_cast<string>(c);
    // Error! Doesn't compile.
 string d = to_string(c);
    // Works!

 string e = "42";
 int f = static_cast<int>(e);
   // Error! Doesn't compile.
 int f = stoi(e);   // string-to-int
   // Works!
   // use stod() for string-to-double
 return 0;
}
```



Statements

- C/C++ functions are composed of statements
- Most common kinds of statements **end with a semicolon**
- Declarations (e.g. `int x=3;`)
- Assignment + Expression (suppose **`int x=3; int y;`**)
  - `x = x * 5 / 9;` // compute the expression & place result in x
    `// x = (3*5)/9 = 15/9 = 1`
- Assignment + Function Call (+ Expression)
  - `x = cos(0.0) + 1.5;`
  - ~~`sin(3.14)`~~`;` // Must save or print out the result (x = sin(3.14), etc.)
- cin, cout statements + Expressions
  - `cout << cos(0.0) + 1.5 << " is the answer." << endl;`
- Return statement (immediately ends a function)
  - `return` *`expression`*`;  // (more on this later)`

Exercises

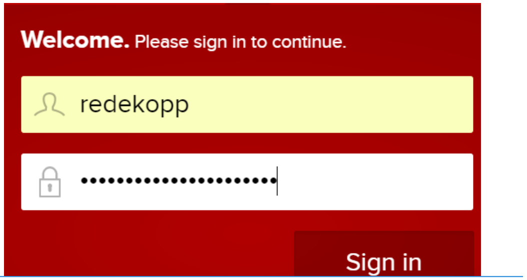
- Exercises:
  - average
  - rad2deg
- Write a program to convert temperature from Celsius to Fahrenheit $[F = \frac{9}{5} \cdot C + 32]$
  - Use http://cpp.sh or http://onlinegdb.com (or EdStem Workspace, if available)

SOLUTIONS

When To Introduce a Variable

- When a value will be input (via cin) and/or change at run-time (as the program executes)

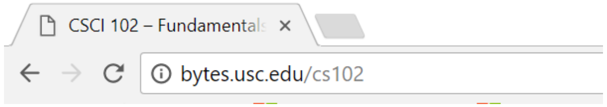
```
string username, password;
cin >> username >> password;
```



Welcome. Please sign in to continue.

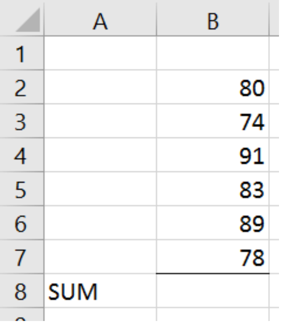
redekopp

••••••••••••••••••••••••



- When a value is computed/updated at one time and used (many times) later

```
int currentSum = 0;
```



	A	B
1		
2		80
3		74
4		91
5		83
6		89
7		78
8	SUM	

- To make the code more readable by another human

```
double a = (x+34) * (n*6.25);

// readability of above vs. below

double height = x + 34;
double width =  n * 6.25;
double area = height * width;
```

What Variables Might Be Needed

- Video playback (YouTube player)
  –

`string url`

`int volume`

`bool fullScreen`

- Calculator App
  –

`double ans`

`char operator`

`double nextValue`

Exercises

- What is printed by the following two programs?

```
#include <iostream>
using namespace std;

int main()
{
  int value = 1;
  value = (value + 5) * (value – 3);
  cout << value << endl;

  double amount = 2.5;
  value = 7;
  amount = value + 6 / amount;
  cout << amount << endl;

  cout << value % 3 << endl;
  return 0;
}
```

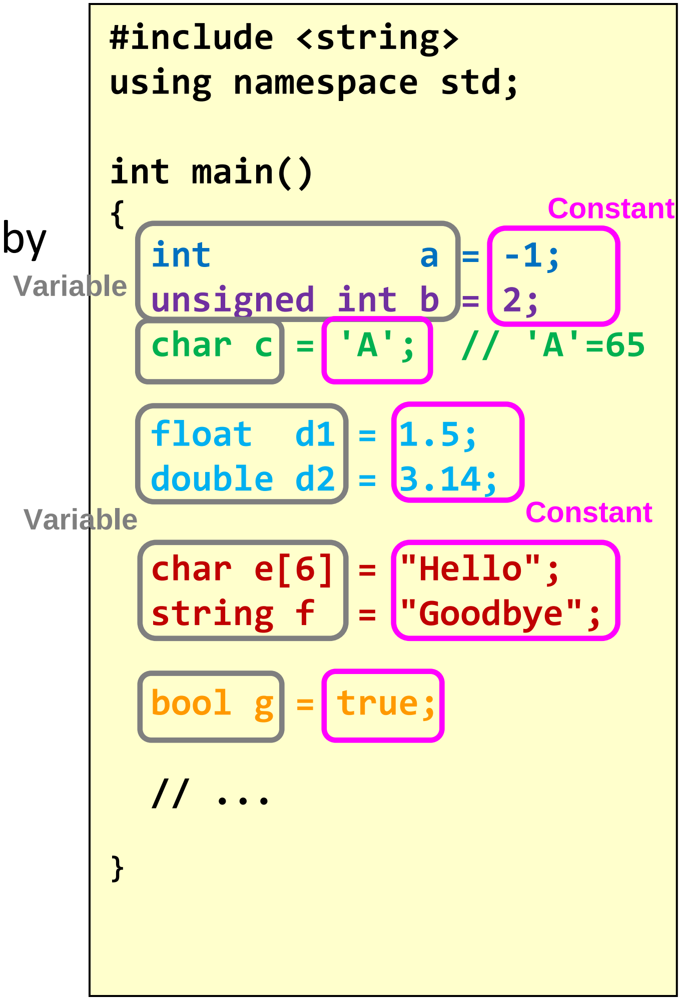
```
-12
9.4
1
```

```
#include <iostream>
using namespace std;

int main()
{
  int x = 5;
  int y = 3;
  double z = x % y * 6 + x / y;

  cout << z << endl;

  z = 1.0 / 4 * (z – x) + y;
  cout << z << endl;

  return 0;
}
```

```
13    // or 13.0
5     // or 5.0
```

C/C++ Variable Types

- A type indicates how many bits / bytes of **storage** (memory) are required and how to **interpret** the number being stored
- **Integer (int) types**
  - Are signed (numbers can be positive or negative) by default, or unsigned (positive-only…including 0)
  - A character (more on this later)
- **Floating point types**: Very large 6.02E23 & very small numbers 6.626E-34)
  - A float or double
- **String/Text types**
  - A single char (1 character)
  - character arrays (C-Strings) / string (preferred…C++ string type)
- **Boolean type**
  - bool (true / false)



```
#include <string>
using namespace std;

int main()
{
  int          a = -1;
  unsigned int b = 2;
  char c = 'A';  // 'A'=65

  float  d1 = 1.5;
  double d2 = 3.14;

  char e[6] = "Hello";
  string f  = "Goodbye";

  bool g = true;

  // ...
}
```

Variable Review: I Do Declare

- (Unlike Python) you must do a **one-time declaration** of a variable before using it
  - Like renting an apartment or storage unit
- If **NOT initialized** via assignment ('='), variables will NOT default to a value like 0, but will contain random data/garbage.
  - Good practice to initialize your variables
- C++ is a **strongly-typed** language which means…
  - You *cannot change* what type of value the variable stores); this is because in C++ a variable name corresponds to a reserved, fixed-size memory location that only fits that specific type

int z;

104	01101000
105	11010001
106	01101000
107	11010001

```cpp
#include <iostream>
using namespace std;
int main() {
  v = 5;       // ERROR: x assigned before
               // it is declared
  int x;       // OK: Declared first but
               //   has random garbage value
  x = 1;       // Need to come back and
               //   initialize later
  int y = 2; // BEST: declare and init.
               //   together
  double z = 3.14; // Good! Declare and init.

  y = "pi is"; // Error: y declared as int
                 // cannot be assigned a string
  y = 5;       // Change value stored in y
  cout << w << " " << y << " " << z << endl;
  return 0;
}
```

C++ is "strongly-typed" and requires variables to be declared before being used.

```python
def main():
    y = 5        # x stores an integer
    z = 3.14
    z = "pi is" # x changes to store a string
    print(x, y)
```

Python does not require explicitly declaring and typing a variable

A Last Note on Variables: Scope

- "Scope" of a variable refers to the
  - **Visibility** (who can access it) and
  - **Lifetime** of a variable (how long is the memory reserved
- For now, there are 2 scopes we will learn
  - **Global:** Variables are declared ***outside*** of any function and are visible to ***all*** the code/functions in the program
    - For various reasons, it is "bad" practice to use global variables. You MAY NOT use them in CS 102.
  - **Local:** Variables are declared ***inside*** of a function and are ***only*** visible in that function and ***die*** when the function ends

```
#include <iostream>
using namespace std;

// Global Variable
int x=1;

int add_x()
{
  int n; // n is a "local" variable
  cin >> n;
  // y and z NOT visible (in scope) here
  // but x is since it is global
  return (n + x);
} // n dies here
int main()
{
  // y and z are "local" variables
  int y=0, z;

  z = add_x();
  y += z / x;  // n is NOT visible
  cout << x << " " << y << endl;
  return 0;
} // y and z die here
```

This Photo by Unknown Author is licensed under CC BY-NC
This Photo by Unknown Author is licensed under CC BY-NC