Week2: Start Programming

Ling Ding

Email: lingding@cs.ucla.edu

Slides modified from Muhao Chen, with permission

Outline

- Variables and Operators
- Undefined Behaviors
- If-else
- While loop
- Input & Output
- Hints for Project 2

Outline

- ***Variables and Operators***
- Undefined Behaviors
- If-else
- While loop
- Input & Output
- Hints for Project 2

Variables and Operators

- A typical variable declaration:

int cs = 400;

- Type:
  - int, double, char, bool, etc..
- E.g.
  - int a = 1;　　　　//a is 1
  - int a = 2;　　　　//a is 2
  - double a = 1;　　　　//a is 1.0
  - double a = 1.1;　　　　//a is 1.1

- **int a = 2.0;　　　　//a == ?**
- //a is 2
- **int a = 2.1;　　　　//a == ?**
- //a is 2

Variables and Operators

- Identifier: Name of variable
  - Starting with letter or underscore
  - Rest part must be letters, digits or underscore
  - Case sensitive
  - Reserved word can not be used

• **int a;**	✔
• **int 2a;**	✘
• **int _a;**	✔
• **int a_;**	✔
• **int while;**	✘
• **int a2;**	✔

- **int apple;**
- **int Apple;**
- **Int APPLE;**

Three different vars

Variables and Operators

To compute tax:

- Programmer 1

```
double cpt_tax(int price)
{
    double tax_rate = 0.1;
    double tax;
    tax = price * tax_rate;
    return tax;
}
```

- Programmer 2

```
double cpt_tax(int price)
{
    double b = 0.1;
    double c;
    c = a * b;
    return c;
}
```

Which one is better?

Name should be chosen carefully, so that program is readable.

Variables and Operators

- What does the assignment operator (i.e. =) do?
  - Assign the value of the right-hand side expression to the left-hand side variable
  - The right-most "=" has highest priority!

```
int x;
int y;
x = 5;
y = 5;
```

```
int x;
int y;
x = (y = 5);
```

```
int x;
int y;
x = y = 5;
```

```
int x = 5;
int y = 5;
```

```
int x=5, y=5;
```

```
int a = 2;
int b = a + 2;   //b is now 4
a = 3;           //b will not change, b is still 4
```

Variables and Operators

- Arithmetic operators (+, -, *, /, %)
  - a = 11 * 3;   //a = 33
  - b = 11(2 + 3);   // error

Variables and Operators

- Compound assignment (+=, -=, *=, /=, %=)
  - **a -= 5;** What does it mean?
  - **a = a - 5**

- Compound assignment examples

count += 2;	**count = count + 2;**
total -= discount;	**total = total - discount;**
bonus *= 2;	**bonus = bonus * 2;**
time /= rushFactor;	**time = time / rushFactor;**
amount *= c1 + c2;	**amount = amount * (c1 + c2);**

Variables and Operators

- Increment Operator: ++
- Decrement Operator: --
- Int i=1;
- i++;

What does i++ mean?
Any equivalent expression?

```
i = i + 1;
i += 1;
```

What about ++i ?

Variables and Operators

- **++i** will increment the value of i, and then return the incremented value.

```
i = 1;
j = ++i;
// i is 2, j is 2
```

```
i += 1;
j = i;
```

- **i++** will increment the value of i, but return the pre-incremented value.

```
i = 1;

j = i++;
// i is 2, j is 1
```

```
j = i;
i +=1;
```

Variables and Operators

- int a=5, b, c;
- b = ++a; **//a is 6 here. b is also 6.**
- c = b++; **//c is 6 here. Then b becomes 7.**

- //What are the values of a, b, c?

a is 6, b is 7, c is 6.

Outline

- *Variables and Operators*
- ***Undefined Behaviors***
- If-else
- While loop
- Input & Output
- Hints for Project 2

Undefined behavior

- What is undefined behavior?
  - In some languages (including C++), the specification leaves the results of certain operation undefined
  - Undefined behavior is often unpredictable and a frequent cause of software bugs.

Undefined behavior

- Divided by zero

```
int a = 10;
int b = a*a;
int c = 25/(b-100);
```

- Overflow

```
int d = 1000;
int e = d*d*d;
int f = e*d;
```

- Uninitialized variables

```
double d;
double e = 2 * d;
cout << e;
```

Undefined behavior

- Uninitialized variables

  what will happen?

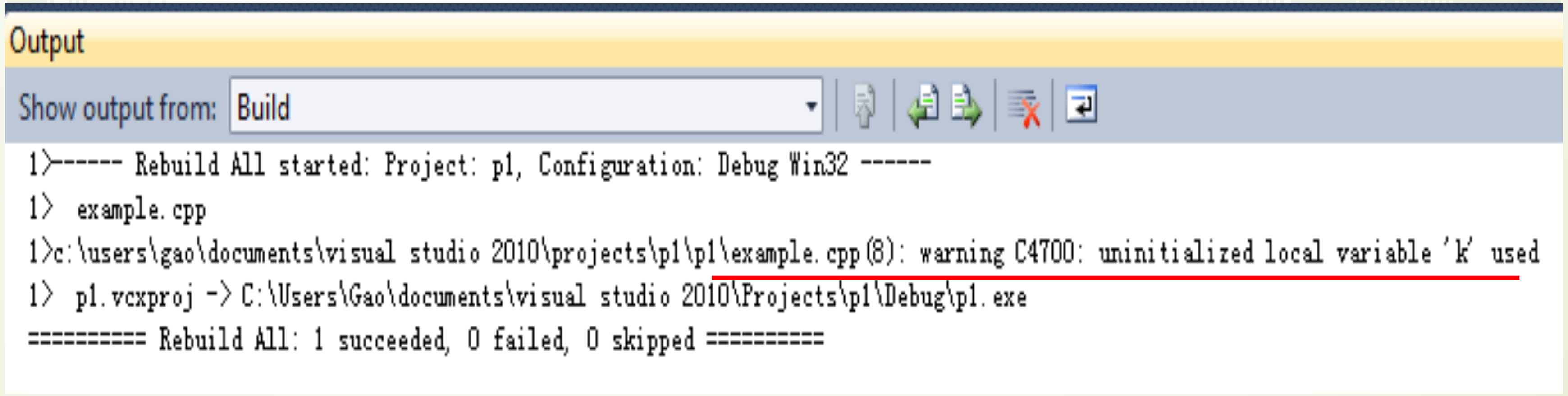
  - might prompt runtime error (Visual C++ debug mode)
  - or might have different values each time you run the program
  - NEVER assume that an uninitialized (int, double) variable will be 0

```
double d;
double e = 2 * d;
cout << e;
```

Undefined behavior

- Uninitialized Variable

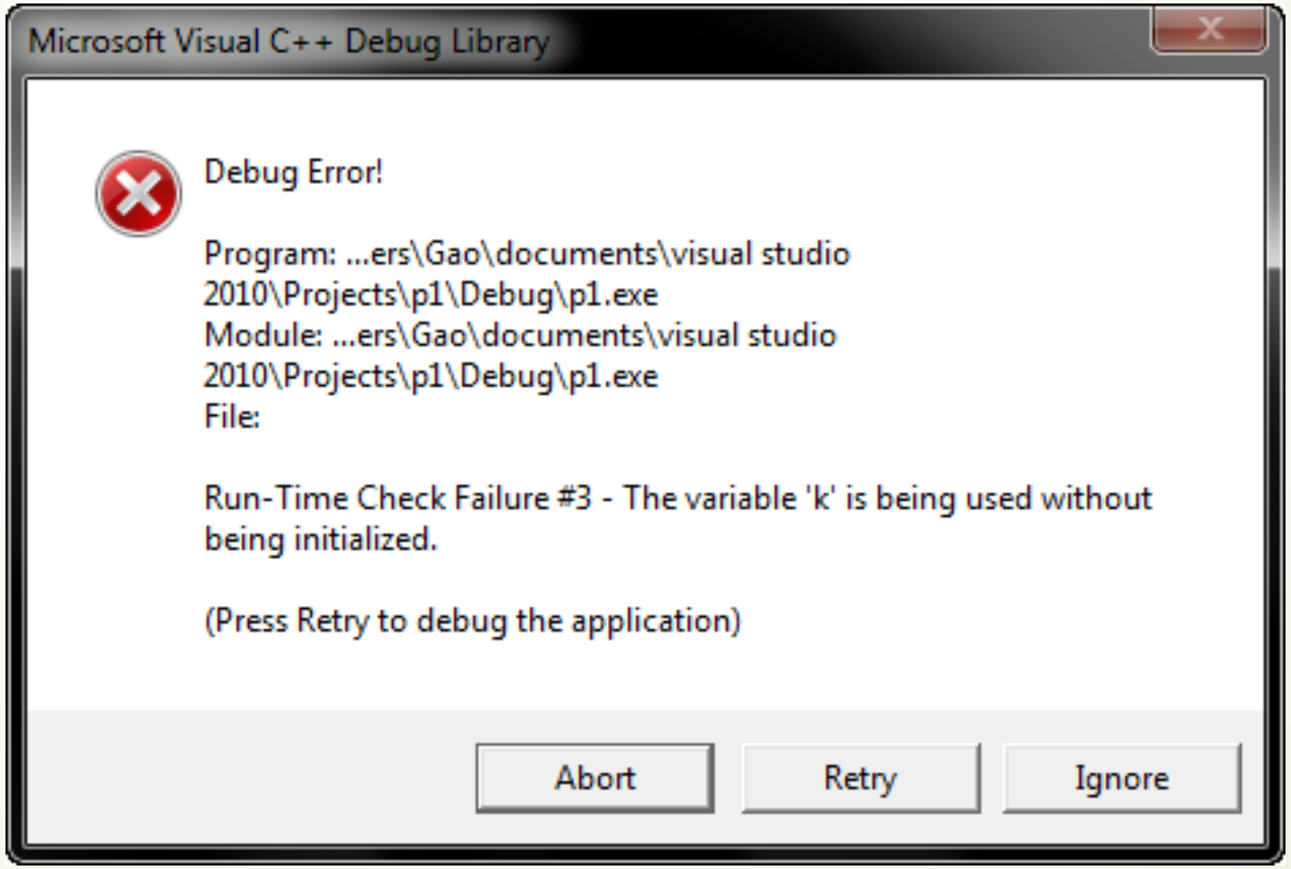
```
#include <iostream>
using namespace std;
int main()
{
  int k;
  cout << k << endl;
}
```



```
1>------ Rebuild All started: Project: p1, Configuration: Debug Win32 ------
1>  example.cpp
1>c:\users\gao\documents\visual studio 2010\projects\p1\p1\example.cpp(8): warning C4700: uninitialized local variable 'k' used
1>  p1.vcxproj -> C:\Users\Gao\documents\visual studio 2010\Projects\p1\Debug\p1.exe
========== Rebuild All: 1 succeeded, 0 failed, 0 skipped ==========
```

Undefined behavior

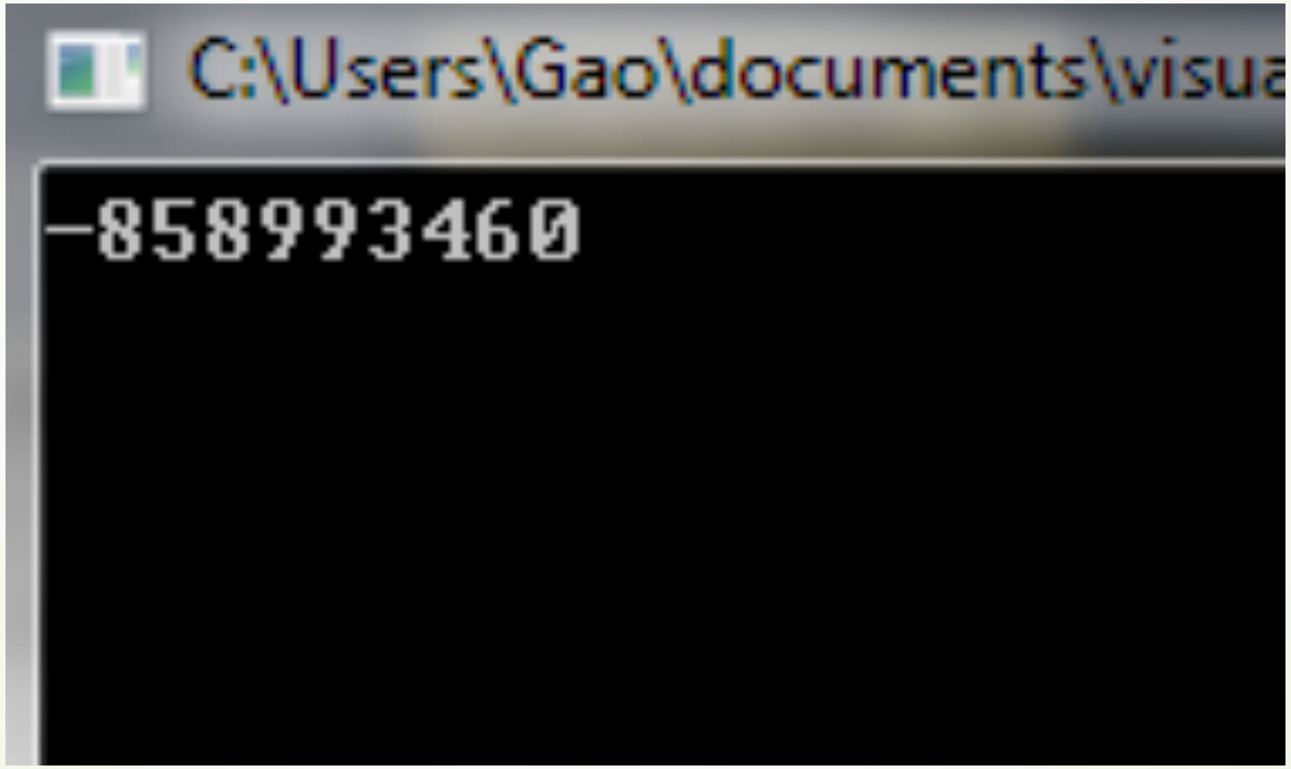
- When you try to run it



```
Debug Error!

Program: ...ers\Gao\documents\visual studio
2010\Projects\p1\Debug\p1.exe
Module: ...ers\Gao\documents\visual studio
2010\Projects\p1\Debug\p1.exe
File:

Run-Time Check Failure #3 - The variable 'k' is being used without
being initialized.

(Press Retry to debug the application)
```

Undefined behavior

- If you really want to see the result …



```
C:\Users\Gao\documents\visua
```

```
-858993460
```

Outline

- *Variables and Operators*
- *Undefined Behaviors*
- ***If-else***
- While loop
- Input & Output
- Hints for Project 2

Comparison Operators

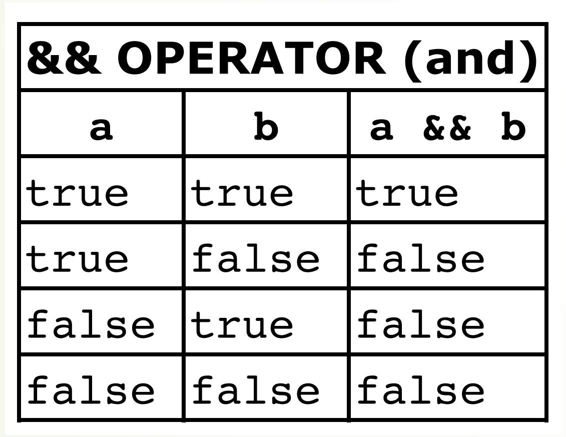
- ==, !=, >=, <=, >, <
- Watch out for boundary cases!
  - If (minutes > 400) or if (minutes >=400) ??
- Watch out for “==“!
  - If (a == 5)
  - If (a = 5) // logical error - might compile, but will do the wrong thing.

Logical Operators

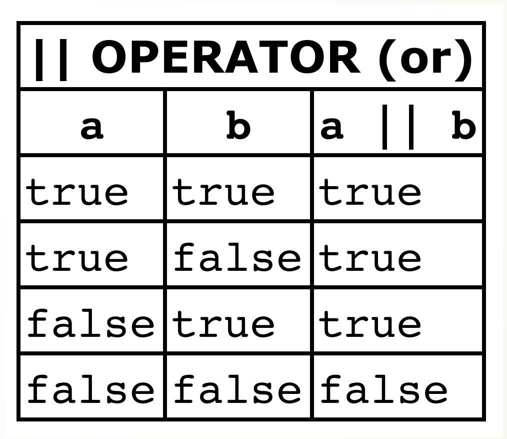
- &&: Boolean logical operation AND
- ||: Boolean logical operation OR
- The logical operators && and || are used when evaluating two expressions to obtain a single relational result.

Logical Operators

- a && b
- a || b



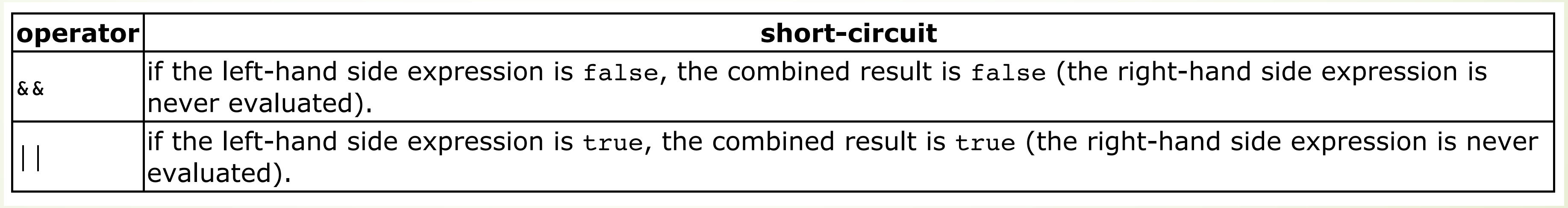
&& OPERATOR (and)		
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false



|| OPERATOR (or)		
a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

Logical Operators

- Short-circuit evaluation
  - When using the logical operators, C++ only evaluates what is necessary from left to right to come up with the combined relational result, ignoring the rest.
  - e.g. (4 > 7) && (8 > 5), it never check whether 8 > 5 is true or not.
  - e.g. (5 == 5) || (3 > 6), it never checks whether 3>6 is true or not.



operator	**short-circuit**
`&&`	if the left-hand side expression is `false`, the combined result is `false` (the right-hand side expression is never evaluated).
`||`	if the left-hand side expression is `true`, the combined result is `true` (the right-hand side expression is never evaluated).

If-else

- When programs need to deal with different conditions in different ways.
  - e.g. If it is sunny tomorrow, I will go swimming, otherwise(else) I stay at home.

  -
    ```
    if (it is sunny tomorrow)
          I go swimming;
    else
          I stay at home;
    ```

If-else

- Format:

```
If (boolean expression)          // bool in parentheses
      statement
else
      statement
```

- Note that the braces “{“ “}” are required when you have multiple statements

If-else

- Need braces “{“ “}” for a block of multiple statements:

```
If (boolean expression) {
      statement 1;
      statement 2;
      …
}
else {
      …
}
```

else is optional

If-else ladder

- Format:

```
If (boolean expression 1)
      statement(s)
else if (boolean expression 2)
      statement(s)
else if (boolean expression 3)
      statement(s)
else
      statement(s);
…
```

Optional

```
cout << “Enter a integer number” << endl;
int x;
cin >> x;

if(x == 0)
  cout << “x is 0.” << endl;
else if(x == 1)
  cout << “x is 1.” << endl;
else
  cout << “x is neither 0 or 1.” << endl;
```

If-else

- Put another if-statement inside a if-statement
- What’s the output?

6

```
int a = b = 4;
if (a == 4)
  if (a == b)
    a++;

if(a!=b)
  a++;
cout << a << endl;
```

If-else

- What’s the output?
  - Compile error

    error: ‘b’ was not declared in this scope
  - b is declared in the if statement, it cannot be seen/used outside the if statement

```
int main()
{
  int a = 4;
  if (a == 4)
    int b = 5;
  cout << b << endl;
}
```

```
int main()
{
  int a = 4;
  if (a == 4) {
    int b = 5;
    cout << b << endl;
  }
}
```

If-else

- A programmer is going to the grocery store and his wife tells him, "Buy a gallon of milk, and if there are eggs, buy a dozen." So the programmer goes, buys everything, and drives back to his house. Upon arrival, his wife angrily asks him, "Why did you get 13 gallons of milk?" The programmer says, "There were eggs!"

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

int main() {
		int milkToBuy = 1;
		bool storeHasEggs;
		cout << “Does the store have eggs ? (1/0) “ ;
		cin >> storeHasEggs;
		if (storeHasEggs)
				milkToBuy += 12;
		cout << “The programmer bought “ << milkToBuy << “ gallons of milk.” << endl;
}
```

Outline

- *Variables and Operators*
- *Undefined Behaviors*
- *If-else*
- ***While loop***
- Input & Output
- Hints for Project 2

While loop

- When a procedure needs to be processed repeatedly.

```
while (boolean expression) {
    statements;
}
```

Loop condition

Loop body

While loop

- A programmer went to the grocery store.
  - His wife said, “while you see watermelons, buy one.”
  - Then he never came back.

```
while (see_watermelons == True) {
        buy_one();
}
come_back();
```

While loop

- Compute factorial(n)

```
unsigned int n, i = 1, result = 1;
cin >> n;
while (i <= n) {
    result *= i++;
}
cout << result;
```

$\longrightarrow$

```
//Another form:
do {
    statements;
} while (boolean expression);
```

While loop

- Two different forms

```
while (boolean expression) {
    statements;
}
```

While: check boolean expression before executing the loop body.

```
do {
    statements;
} while (boolean expression);
```

Do … while: execute the loop body before checking the boolean expression.

- Differences?

while... and do ... while

```
int main()
{
  int n=3, i=4, result=1;
  while (i <= n) {
      result *= i;
      i++;
  }
  cout << result;
}
```

- n=3, i=4
- **i <=n is false. do not start loop**
- result is 1

Check the loop condition before exec loop body

```
int main()
{
  int n=3, i=4, result=1;
  do {
      result *= i;
      i++;
  } while (i <= n);
  cout << result;
}
```

- n=3, i=4
- execute loop body first. result is 1*4=4. i++;   // i becomes 5
- **i <=n is false. End loop**
- result is 4

Exec loop body before checking the loop condition

While loop

- **Watch out for dead loop.**

```
int n=3, i=1, result=1;
  while (i <= n) {
      result *= i;
      i++;
}
```

Make sure that the loop must reach some condition to jump out

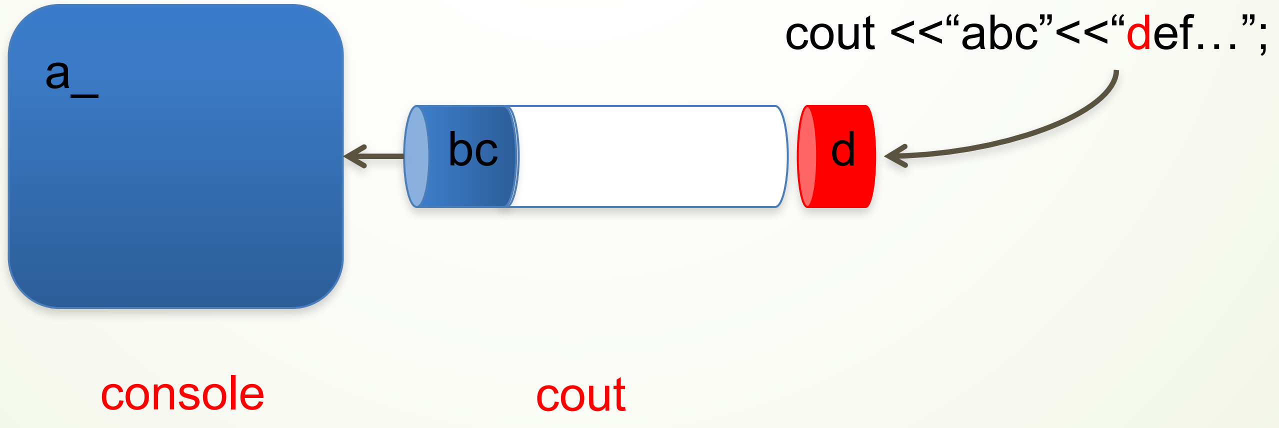
Outline

- *Variables and Operators*
- *Undefined Behaviors*
- *If-else*
- *While loop*
- ***Input & Output***
- Hints for Project 2

Input & Output

- std::cin standard input stream
- std::cout standard output stream
- connect your IO to devices.

e.g. cout << "abcdef…";
equivalent to: cout << "abc" << "def…";



Output

- Output types:
  - String
  - Integer
  - float/double
  - …
  - e.g.

```
int dog_num = 3;
float weight = 4.5;
cout << “I have ” << dog_num << “ dogs, their average weight is ”
   << weight << “lbs.” <endl;
```

Input

```
string s;
int i;
```

- **getline(cin, s);**
  - consumes all the characters available up to and including a newline
  - throw away the newline
  - stores the other characters in s
- **cin >> i;**
  - consumes and discards any leading whitespace (blanks, tabs, and even newlines)
  - consumes an optional minus sign and a sequence of digits
  - DOES NOT consume the character after the last digit
  - sets i to the number represented

Input

- You can specify different types to input in a sequence, e.g.:

```
int a;
double b;
cin >> a;
cin >> b;
string s;
getline(cin, s);
```

input:

```
3
4.5
Hello
```

How to input a string after numerical numbers?

- Use getline(cin (s) to input string!

What's the value of s?

Input – String use getline

- **cin.ignore(10000, '\n');**
  - consume and discard either all characters up to the next newline or 10000 characters, whichever comes first.
  - We pick a huge number so that the latter situation will never occur
  - Notice that we use single quotes, not double quotes, to denote the single newline character here

Input – String use getline

- **cin.ignore(10000, '\n');**

The **only** time to use cin.ignore(...) is after you feed a numerical variable with cin and use getline(cin, s) to feed a string, i.e.:

```
int i;
string s;
cin>> i;
cin.ignore(10000, ’\n’);
getline(cin, s);
```

Input

- Test of a string is empty
  - empty string: “”
  - string with only blanks: “ “ or “   “

Empty string is not the same as string with only blanks!

```
cout << "What is your name? ";
string name;
getline(cin, name);
if (name == "")
        cout << "You didn't type a name!" << endl;
else
        cout << "Hello, " << name << endl;
```

Input – String use getline

```
Please input your UCLA ID: 123456789
Please input your name: Ling Ding
Input data: 123456789 Ling Ding
```

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

int main()
{
        int UID;
        string name;
        cout << "Please input your UCLA ID: " ;
        cin >> UID;
        cout << "Please input your name: ";
        cin.ignore(10000, '\n');
        getline(cin, name);
        cout << "Input data: " << UID << " " << name << endl;
}
```

Floating Numbers

- Why do we use floating point numbers?
  - To store numbers with a fractional component
  - To store very large numbers
- Three different floating point data types:

Type	Size	Range
float	4bytes	$-3.4*10^{38} \sim 3.4*10^{38}$ (7decimal digits)
double	8bytes	$-1.7*10^{304} \sim 1.7*10^{304}$ (16decimal digits)
long double	8bytes or 16bytes	varies

Floating Numbers

- How to assign values to floating point variables?

  double dValue = 500.0;

- Other ways?

  double dValue = 5e2;

  //Assign 500 to dValue using scientific notations

- Another example:

  double dValue = 0.05;

  double dValue = 5e-2;

Floating Numbers

- How to transform integers to floating point variables

```
int a = 10, b = 8;
```

a/b: an integer

double c = (double) a / b;	1.25
double c = (double) (a / b);	1.0
double c = 1.0 * a / b;	1.25
double c = a / (double) b;	1.25
double c = a / b * 1.0;	1.0
double c = a/ (b * 1.0);	1.25

Outline

- *Variables and Operators*
- *Undefined Behaviors*
- *If-else*
- *While loop*
- *Input & Output*
- ***Hints for Project 2***

Project 2

- Incremental development
  - Break the task down into several sub-goals!
  - Testing each sub-goal before moving on to the next, e.g.:
    1. Define vars and get input
    2. Clarify the if-else logic (What to do under each condition?)
       1) Case 1
       2) Case 2
       3) …
    3. Output

Project 2

- Programming project part of Project 2 DOES NOT use any loops. (The homework part does)
- How to get string input with **getline(...)**
- When to use **cin.ignore(...)**
- Watch out for boundary cases!

Thank you!