Introduction to Programming (CS 101)

Spring 2024

Lecture 6:

Internal representations of data types

Instructor: Preethi Jyothi

Based on material developed by Prof. Abhiram Ranade and Prof. Manoj Prabhakaran

Reminder: Theory Quiz 1

- **Details**:
  - Date/ Time: Feb 5th, 2025, 8:30 am to 9:30 am
  - Venues: LA001, LA002, LA201, LA202, LH101, LH102

- **Syllabus**:
  - All the content covered in lectures 1 to 5

- **Mode**:
  - Closed book / closed notes quiz
  - No devices (phones, laptops, tablets, smart watches, etc.) allowed
  - No calculator needed
  - Seating chart will be shared next week

Recap-I: To increment or not to increment (and continue)

What is the output of the following program?

2,1

1,1

0,2
1,1

0,1
1,1

1,2

```
main_program {

 for(int i = 0; i < 2; i++) {
   for(int j = 0; j < 2; j++) {
     if(i == j++)
        continue;
     cout << i << "," << j << endl;
   }
 }

}
```

Recap-II: Compute n choose 2

Given a non-negative integer n, fill in all three boxes in the code template below to compute $\binom{n}{2} = C(n,2) = \frac{n!}{2(n-2)!}$ (the well-known combination formula to choose 2 from n items)

```
main_program {
  unsigned int n, count = 0;
  cin >> n;

  for(int i = 1;        ;i++) {
     for(int j =       ;       ;j++)
        count += 1;
  }

  cout << n << " choose 2 = " << count << endl;
}
```

Recap-II: Compute n choose 2

Given a non-negative integer n, fill in all three boxes in the code template below to compute $\binom{n}{2} = C(n,2) = \frac{n!}{2(n-2)!}$ (the well-known combination formula to choose 2 from n items)

```
main_program {
  unsigned int n, count = 0;
  cin >> n;

  for(int i = 1; i < n; i++) {
     for(int j = i + 1; j <= n; j++)
        count += 1;
  }

  cout << n << " choose 2 = " << count << endl;
}
```

Data types and their representations

CS 101, 2025

`bool` type

- Simplest data type that takes two values `true`, `false`.
  - Internally occupies a ***byte*** of memory (equivalent to ***8 bits***)

- Other types can be converted to `bool`
  - Implicitly: By using it where a `bool` expression is expected. E.g., `bool b = expr`, `if(expr)`, etc.
  - Explicitly: An expression `expr` can be explicitly cast to `bool` by writing `bool(expr)`

- Zero is treated as `false`, all other non-zero values are treated as `true`
  - Conversely, for `bool` to `int`, `false` converted to `0` and `true` to `1`
  - Arithmetic on `bool` treats the underlying quantities like integers

- What is `x` here?

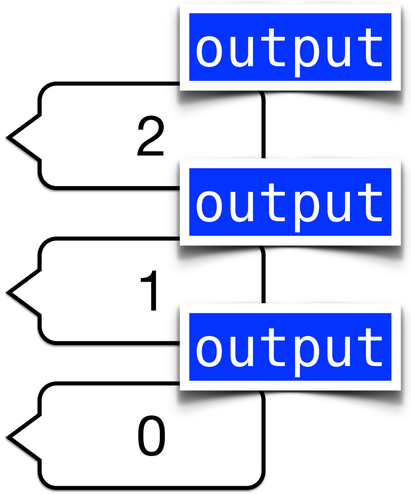
```
int x; bool y; x = y = 5;
```

output

x = 1

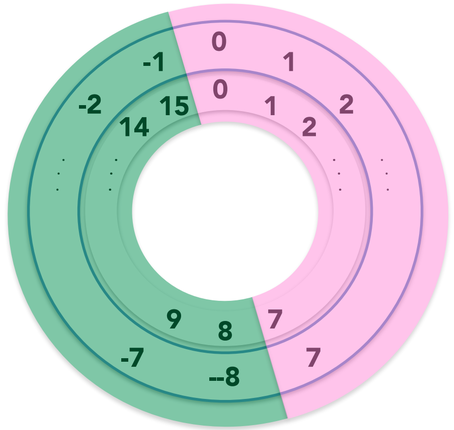
`bool` arithmetic

```
bool b1, b2;
b1 = 1; b2 = -1;

cout << b1 + b2 << endl;

 cout << (b1+=b2) << endl;

 cout << (b1+=-1) << endl;
```



Binary representation

- Using n bits, we can represent 2^n different numbers
  - $b_{n-1} \ldots b_1 b_0$ stands for the number $b_{n-1}*2^{n-1} + \ldots + b_1*2^1 + b_0*2^0$
  - Numbers [0,7] uses 3 bits, [0,15] uses 4 bits
- We can use bits to represent negative numbers too
- Standard format: Two's complement
  - n bits to represent integers in the range $[-2^{n-1}, 2^{n-1} - 1]$



0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
-8	8	1000
-7	9	1001
-6	10	1010
-5	11	1011
-4	12	1100
-3	13	1101
-2	14	1110
-1	15	1111

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

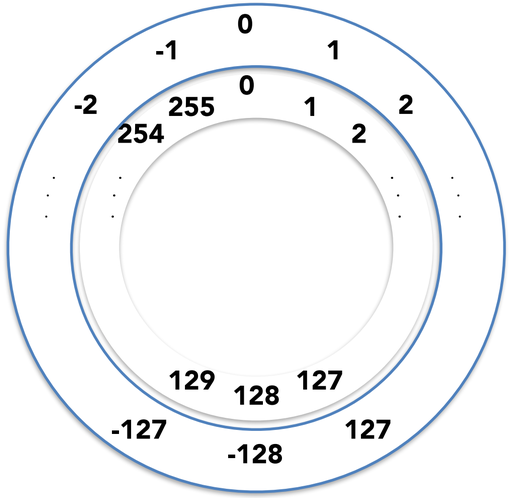
0	00
1	01
2	10
3	11

0	0
1	1

`char` type

char	ASCII
0	48
1	49
2	50
3	51

- In C++, `char` data type corresponds to a single byte, i.e., `8` bits
- unsigned char works like an integer in the range `[0,255]`
  - `00000000` is `0`, and `11111111` is `255`
- signed char works like an integer in the range `[-128, 127]`
  - `00000000` is `0`, and `01111111` is `127`
  - `10000000` is `-128`, and `11111111` is `-1`



- Operations like `+, -, *, /` work with `char` like for integers, and the result is an integer (can be converted back to `char`, if desired)
- Converting (implicitly or by casting) integers to `char` is done by mod 256 (i.e. `% 256`)

```
int x = 306; char y; y = x; cout << y;
```

2 output

Bitwise operations

- We can carry out bit-level manipulations using `&`, `|`, `~` bitwise operators: Same semantics as AND, OR, NOT, respectively

- These operators can be applied **bitwise** on bytes
  - `00001111 & 11110000 → 00000000`
  - `00001111 | 11110000 → 11111111`
  - `~00110011 → 11001100`

- XOR operator, `^`:
  - `0 ^ 0 → 0`
  - `1 ^ 0 → 1`
  - `0 ^ 1 → 1`
  - `1 ^ 1 → 0`

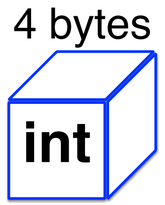
Example: `char` operations

- Demo in class of code to convert lowercase to uppercase and vice-versa

- Note:
  - ASCII code of the character '`A`' is `01000001 (65)`
  - ASCII code of the character '`a`' is `01100001 (97)`

`int` type

- `int` and `unsigned int` correspond to `4` bytes, i.e., `32` bits
- `int` takes values in the range $[-2^{31}, 2^{31}-1]$ (≈ [-2 billion, 2 billion])
- `unsigned int` takes values in the range $[0, 2^{32} - 1]$ (≈ [0, 4 billion]
- `short int` and `unsigned short int` correspond to 16 bits
- `long long int` and `unsigned long long int` correspond to 64 bits
- `long int` and `unsigned long int` correspond to 32 bits (same as `int`)



Next class: More about data types

CS 101, 2025