IO

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Input and Output

Your Objectives:

- Write input routines for three kinds of test inputs,
- use ‘cin‘, ‘scanf‘, and ‘printf‘ properly for various types of variables, and
- write code for interactive tests.

Explicit Test Count

- First line of input is the number of tests you will receive.

```c
#include <stdio.h>

int main() {
  int cases,x,y;
  scanf("%d",&cases);
  while (cases>0) {
    --cases;
    scanf("%d %d",&x,&y);
    printf("%d\n",x+y);
  }
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);

  int cases,x,y;
  cin >> cases;
  while (cases>0) {
    --cases;
    cin >> x >> y;
    cout << x + y << "\n";
  }
```

Termination Marker

- The input itself will use a special value.

```c
#include <stdio.h>

int main() {
  int x,y;
  while (1) {
    scanf("%d %d",&x,&y);
    if (x==-1 && y==-1)
        break;
    printf("%d\n",x+y);
  }
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);
  int x,y;
  while (1) {
    cin >> x >> y;
    if (x==-1 && y==-1)
        break;
    cout << x + y << "\n";
  }
}
```

Termination Marker, pt 2

```
#include <stdio.h>

int main() {
  int x,y;
  while (scanf("%d %d",&x,&y) && x != -1 && y != -1) {
    printf("%d\n",x+y);
  }
}
```

- A similar trick works with `cin`.

End of File

- Use EOF explicitly.

```c
#include <stdio.h>

int main() {
  int x,y;
  while (scanf("%d %d",&x,&y) != EOF) {
    printf("%d\n",x+y);
  }
}
```

- Use cin.eof() for cin.

Why scanf and printf?

- There are problems that TLE if you use `cin` and `cout`.
  - This happens if the problem requires a *lot* of output.
  - Use `ios_base::sync_with_stdio(false);` and `cin.tie(NULL);` to prevent flushing the output. (Maybe put that in your TRD!)
  - Similarly, `endl` will force the output to be flushed. Use `\n` instead.
- `scanf` has some regular-expression like features that can be useful.
- On the other hand, you must match the type and pass in a reference.

Code	**Meaning**
`%d`	Scan an integer
`%lld`	Scan a long long integer
`%s`	Scan a string
`%c`	Scan a character

Spaces and such

- Literal Characters

```c
// will read "(10,20)"
scanf("(%d,%d)");
```

- Spaces

```c
// will read "(10,20)", " ( 10, 20 )", but not "(10 ,20)"
scanf(" ( %d, %d )");
```

- A binary followed by vowels

```c
// will read "110101 eieio"
scanf("%[01] %[aeiou]");
```

Getting a whole line

- `fgets (name, 100, stdin);` will read a whole line into the string.
- `getline(cin, name);`
- `getline(cin >> std::ws, name);` to read leading whitespace first.
- Sometime you will need to parse out the line yourself afterward; this can be tricky. Avoid it if possible.

Setting number of digits.

- For `printf` codes:
  - `%d` output an integer
  - `%5d` output an integer, using 5 characters, leading spaces.
  - `%05d` output an integer, using 5 characters, leading zeros.
  - Similar codes exist for floating point.

  ```
  int n = 25;
  printf("%d, %5d, %05d\n",n,n,n);
  ```

  Output: `25,    25, 00025`
- For `cout`, use e.g., `cout << width(5) << n << "\n";`

Interactive Tests

- Not common yet, but ICPC is starting to use them.
- Opposite advice now applies: call `flush(stdout)` every time you print, or else use `endl` with `cout`.

```
#include <stdio.h>

int main() {
  int x,y;
  while (scanf("%d %d",&x,&y) != EOF) {
    if (x==-1 && y==-1)
        break;
    printf("%d\n",x+y);
    fflush(stdout);
  }
}
```