Reading input in C

Adapted from materials by Dr. Carrier

Rishi - Unsplash

Where do we get input?

Where do we get input?

- stdin
- Command line arguments
- Files

Where do we get input?

- **stdin** (our focus here)
- Command line arguments
- Files

scanf

scanf

```
scanf(format_str, mem_addr_1, mem_addr_2, …);
```

scanf

```
scanf(format_str, mem_addr_1, mem_addr_2, …);
```

- format_str is a formatted string like in printf
  - E.g., “%d %f” for an int, space, float

scanf

```
scanf(format_str, mem_addr_1, mem_addr_2, …);
```

- format_str is a formatted string like in printf
  - E.g., “%d %f” for an int, space, float
- Each specified var (e.g., %d), will need a memory address

scanf

```
scanf(format_str, mem_addr_1, mem_addr_2, …);
```

- format_str is a formatted string like in printf
  - E.g., “%d %f” for an int, space, float
- Each specified var (e.g., %d), will need a memory address
  - Remember, it’s a pointer so the function can change the underlying memory!

scanf

```
scanf(format_str, mem_addr_1, mem_addr_2, …);
```

- format_str is a formatted string like in printf
  - E.g., “%d %f” for an int, space, float
- Each specified var (e.g., %d), will need a memory address
  - Remember, it’s a pointer so the function can change the underlying memory!
- scanf returns a EOF (constant) if EOF (Ctrl+D) is sent via stdin

scanf

```
scanf(format_str, mem_addr_1, mem_addr_2, …);
```

- format_str is a formatted string like in printf
  - E.g., “%d %f” for an int, space, float
- Each specified var (e.g., %d), will need a memory address
  - Remember, it’s a pointer so the function can change the underlying memory!
- scanf returns a EOF (constant) if EOF (Ctrl+D) is sent via stdin
  - We can use this to know when the user wants to stop

scanf

```
scanf(format_str, mem_addr_1, mem_addr_2, …);
```

- format_str is a formatted string like in printf
  - E.g., “%d %f” for an int, space, float
- Each specified var (e.g., %d), will need a memory address
  - Remember, it’s a pointer so the function can change the underlying memory!
- scanf returns a EOF (constant) if EOF (Ctrl+D) is sent via stdin
  - We can use this to know when the user wants to stop

Documentation:
https://cplusplus.com/reference/cstdio/scanf/

scanf - example

`scanf(format_str, mem_addr_1, mem_addr_2, …);`

```
int x = 0;
int sum = 0;
while(1){
   int res = scanf(“%d”, &x);
   if(res == EOF) break;
   sum += x;
}
printf(“Sum is %d\n”, sum);
```

fgets

```
fgets(char* s, int size,FILE* stream);
```

fgets

```
fgets(char* s, int size,FILE* stream);
```

- For now, our stream is stdin (built in)

fgets

```
fgets(char* s, int size,FILE* stream);
```

- For now, our stream is stdin (built in)
- We need to allocate char buffer

fgets

```
fgets(char* s, int size,FILE* stream);
```

- For now, our stream is stdin (built in)
- We need to allocate char buffer
- Will read, at most, n-1 chars
  - Null terminator (\0) placed after last char read

fgets

```
fgets(char* s, int size,FILE* stream);
```

- For now, our stream is stdin (built in)
- We need to allocate char buffer
- Will read, at most, n-1 chars
  - Null terminator (\0) placed after last char read
- Stops at newline or EOF

fgets

```
fgets(char* s, int size,FILE* stream);
```

- For now, our stream is stdin (built in)
- We need to allocate char buffer
- Will read, at most, n-1 chars
  - Null terminator (\0) placed after last char read
- Stops at newline or EOF
- Returns s if successful, NULL if not

fgets

```
fgets(char* s, int size,FILE* stream);
```

- For now, our stream is stdin (built in)
- We need to allocate char buffer
- Will read, at most, n-1 chars
  - Null terminator (\0) placed after last char read
- Stops at newline or EOF
- Returns s if successful, NULL if not

Documentation:

https://cplusplus.com/reference/cstdio/fgets/

fgets - example

```
fgets(char* s, int size,FILE* stream);
```

```
char s[10];
int main(){
  while(1){
     char* res = fgets(s, 10, stdin);
     printf("received input: %s\n", s);
     if(res == NULL) break;
  }
}
```

getline

```
getline(char** lineptr, size_t *n,FILE* stream);
```

getline

```
getline(char** lineptr, size_t *n,FILE* stream);
```

- For now, our stream is stdin (built in)

getline

```
getline(char** lineptr, size_t *n,FILE* stream);
```

- For now, our stream is stdin (built in)
- Not in C standard, part of POSIX > 2008

getline

```
getline(char** lineptr, size_t *n,FILE* stream);
```

- For now, our stream is stdin (built in)
- Not in C standard, part of POSIX > 2008
- Delimits by newline

getline

```
getline(char** lineptr, size_t *n,FILE* stream);
```

- For now, our stream is stdin (built in)
- Not in C standard, part of POSIX > 2008
- Delimits by newline
- Reallocates memory if more room is needed

getline

```
getline(char** lineptr, size_t *n,FILE* stream);
```

- For now, our stream is stdin (built in)
- Not in C standard, part of POSIX > 2008
- Delimits by newline
- Reallocates memory if more room is needed
- Allocates memory for you if lineptr is NULL and n=0
  - Updates both values

getline

```
getline(char** lineptr, size_t *n,FILE* stream);
```

- For now, our stream is stdin (built in)
- Not in C standard, part of POSIX > 2008
- Delimits by newline
- Reallocates memory if more room is needed
- Allocates memory for you if lineptr is NULL and n=0
  - Updates both values
- Returns the number of chars read
  - Or -1 for errors

getline

```
getline(char** lineptr, size_t *n,FILE* stream);
```

- For now, our stream is stdin (built in)
- Not in C standard, part of POSIX > 2008
- Delimits by newline
- Reallocates memory if more room is needed
- Allocates memory for you if lineptr is NULL and n=0
  - Updates both values
- Returns the number of chars read
  - Or -1 for errors

Documentation:
https://man7.org/linux/man-pages/man3/getline.3.html

getline - example

```
getline(char** lineptr, size_t *n,FILE* stream);
```

```
void main(){
   while(1){
      char* s = NULL;
      size_t n = 0;
      int res = getline(&s, &n, stdin);
      printf("Received: %s\n", s);
      free(s);
      if(res == -1) break;
   }
}
```

Notes

- Be careful about newlines
  - scanf(“ %c”, …) is recommended over scanf(“%c\n”, …)
  - This is why printing strings from fgets and getline have empty lines after them!
- Be careful mixing and matching these functions
  - e.g., fgets and getline consume newlines
    - scanf does not
- I prefer scanf
  - It makes dealing with non-string types easier
    - We’ll talk later about str->int conversion