1. Command Line Utilities for Argument Parsing

File: `univ_cmdutils.h`
Author: Soham Metha
Date: January 2025

The `univ_cmdutils.h` header provides generic utility functions for handling command line arguments efficiently.

1.1. Table of Contents

- 1. Command Line Utilities for Argument Parsing
  - 1.1. Table of Contents
  - 1.2. Data Structures
    - 1.2.1. Option Enumeration
  - 1.3. Functions
    - 1.3.1. getNextCmdLineArg
      - 1.3.1.1. **Declaration**
      - 1.3.1.2. **Description**
      - 1.3.1.3. **Parameters**
      - 1.3.1.4. **Return Value**
      - 1.3.1.5. **Usage Example**
    - 1.3.2. flagAsOption
      - 1.3.2.1. **Declaration**
      - 1.3.2.2. **Description**
      - 1.3.2.3. **Parameters**
      - 1.3.2.4. **Return Value**
      - 1.3.2.5. **Usage Example**
  - 1.4. Details
    - 1.4.1. Internal Mapping for Flags and Options
    - 1.4.2. Example Usage
  - 1.5. Notes

1.2. Data Structures

1.2.1. Option Enumeration

The `Option` enumeration defines a set of possible command line options that correspond to specific flags:

Value	Description
`FILE_INPUT`	Represents an option for file input.
`FILE_OUTPUT`	Represents an option for file output.

Value	Description
`MODE_ASSEMBLE`	Represents assembly mode.
`MODE_DISASSEMBLE`	Represents disassembly mode.
`ASM_LANG`	Represents assembly language.

1.3. Functions

1.3.1. getNextCmdLineArg

1.3.1.1. Declaration

```
char* getNextCmdLineArg(int* argc, char*** argv);
```

1.3.1.2. Description

This function retrieves the next argument from the command line argument list (`argv`) and updates the argument count (`argc`) and argument list pointers to reflect the remaining arguments.

1.3.1.3. Parameters

Parameter	Description
`argc`	A pointer to the number of remaining arguments.
`argv`	A pointer to the list of argument strings.

1.3.1.4. Return Value

- Returns the next command line argument as a string.

1.3.1.5. Usage Example

```
int main(int argc, char* argv[]) {
    char* arg = getNextCmdLineArg(&argc, &argv);
    printf("Program Name : %s\n", arg);
    return 0;
}
```

1.3.2. flagAsOption

1.3.2.1. Declaration

```
Option flagAsOption(char* s);
```

1.3.2.2. Description

This function maps a command line flag (e.g., `-i`, `-o`) to its corresponding `Option` enumeration value. It uses an internal mapping (`OptionStringMap`) to perform the conversion.

1.3.2.3. Parameters

Parameter	Description
`s`	The command line flag string to be converted.

1.3.2.4. Return Value

- Returns the corresponding `Option` enumeration value if the flag is found.
- Returns `-1` if no matching flag is found.

1.3.2.5. Usage Example

```
Option opt = flagAsOption("-i");
if (opt == FILE_INPUT) {
    printf("File input option selected.\n");
}
```

1.4. Details

1.4.1. Internal Mapping for Flags and Options

The following mapping is used internally to convert command line flags to their corresponding `Option` enumeration values:

```
-   `"-i"` → `FILE_INPUT`
-   `"-o"` → `FILE_OUTPUT`
-   `"-a"` → `MODE_ASSEMBLE`
-   `"-d"` → `MODE_DISASSEMBLE`
-   `"-l"` → `ASM_LANG`
```

1.4.2. Example Usage

Here's how you can use the provided utilities in a typical command line program:

```c
#include "univ_cmdutils.h"
#include <stdio.h>

int main(int argc, char* argv[]) {
    while (argc > 0) {
        char* arg = getNextCmdLineArg(&argc, &argv);
        processFlag(arg, &argc, &argv);
    }
    return 0;
}
void processFlag(char* flag, int* argc, char*** argv)
{

    Option opt = flagAsOption(flag);

    switch (opt) {
    case FILE_INPUT:
        printf("Option: File Input\n");
        break;
    case FILE_OUTPUT:
        printf("Option: File Output\n");
        break;
    case MODE_ASSEMBLE:
        printf("Option: Assemble Mode\n");
        break;
    case MODE_DISASSEMBLE:
        printf("Option: Disassemble Mode\n");
        break;
    case ASM_LANG:
        printf("Option: Assembly Language\n");
        break;
    default:
        printf("Unknown or invalid flag: %s\n", arg);
        break;
    }
}
```

1.5. Notes

- Ensure that the flags provided match the ones defined in the `OptionStringMap`.
- Be cautious with memory management for dynamically allocated arguments or strings.
- This implementation assumes valid input and does not handle malformed command line arguments robustly.