TP WEB SCRAPPING

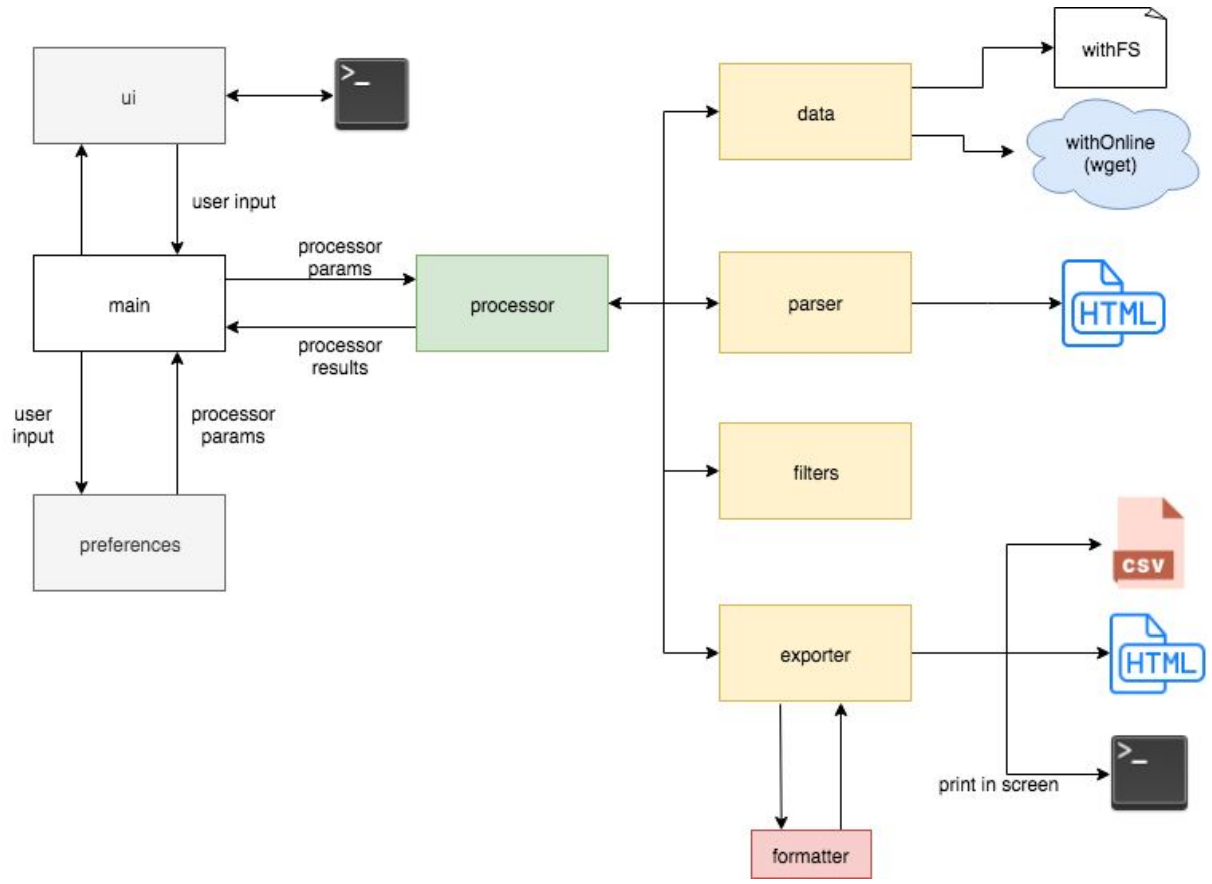
Integrantes

Martín Morales
Pablo Cha

Cursada: K2054 Jueves noche

Repo:
https://github.com/tinchotricolor22/SSL-TP-Bolsar

Arquitectura



- **main**: punto de entrada. Se encarga de llamar a la ui, obtener las preferences para el reporte que eligió el usuario y con ellas llamar al processor
- **processor**: es el flujo principal. Se encarga de manejar el input de la respuesta entre componentes.
- **data**: obtiene el FILE* con el método que se elija (FS o WGET).
- **parser**: parsea el archivo a un array de struct de dominio.
- **filters**: filtra información del array dependiendo implementando lógica de negocio que se provea.
- **exporter**: recibe la información y llama al método de exportación seleccionado, aplicando los formatos correspondientes-
- **formatter**: aplica un formato condicional a un componente, dependiendo la lógica de negocio que se provea.

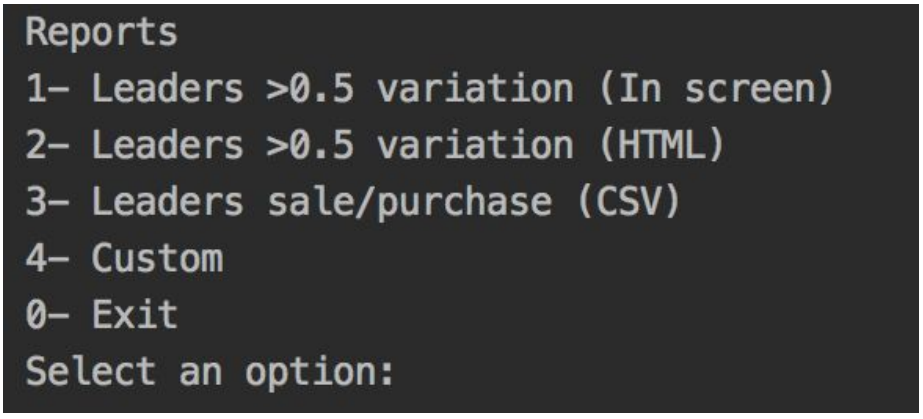
En la stuct de dominio nos referimos a leader.h, donde tenemos un estructura para poder manejar durante el flujo del programa.

El programa se probó en macOS, pero debería funcionar en sistemas windows u otra distro de linux. Hay una directiva de precompilación que identifica el SO.

Manual de usuario

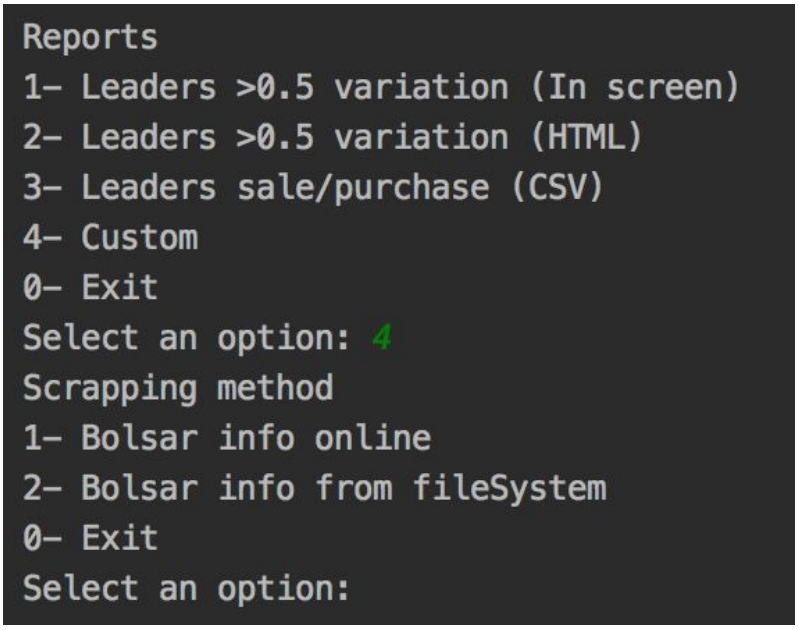
Siempre que se elija método online, se va a obtener la data de
http://52.67.191.9/SSL/lideres-bcba_limpio.html

Pantalla de selección de reporte:



1 - Busca la información **online** y muestra en pantalla las especies que tienen variación > 0.5
2 - Busca la información **online** y exportar un html en la ruta ***output/lideres_bcba_result.html*** las especies que tienen variación > 0.5
3 - Busca la información **online** y exportar un csv en la ruta ***output/lideres_bcba_result.csv*** los datos de la tabla del líderes.
4 - Nos lleva a la siguiente pantalla, donde podemos seleccionar el input de donde obtener la información.
0 - Sale del programa

Pantalla del método de scrapping



1 - Busca la información **online**
2 - Busca la información en el **file system** , en el path ***/resoureces/lideres_limpio.html***
0 - Sale del programa

Las opciones 1 y 2 nos llevan a la siguiente pantalla, donde podemos seleccionar el método de exportación.

Pantalla del método de exportación

```
Reports
1- Leaders >0.5 variation (In screen)
2- Leaders >0.5 variation (HTML)
3- Leaders sale/purchase (CSV)
4- Custom
0- Exit
Select an option: 4
Scrapping method
1- Bolsar info online
2- Bolsar info from fileSystem
0- Exit
Select an option: 1
Export types
1- CSV
2- HTML
3- Print in screen
0- Exit
Select an option:
```

1 - Exporta un csv en la ruta ***output/lideres_bcba_result.csv***
2 - Exporta un html en la ruta ***output/lideres_bcba_result.html***
3 - Muestra la información por pantalla
0 - Sale del programa

Códigos de respuesta

El main, luego de hacer el procesamiento, puede devolver distintos códigos. Esto son:

- 0 - OK
- 1 - ERROR OBTENIENDO LA DATA
- 2 - ERROR PARSEANDO EL ARCHIVO
- 3 - ERROR FILTRANDO EL ARCHIVO
- 4 - ERROR EXPORTANDO EL ARCHIVO

Conclusiones

Si bien el scope de la consigna no lo decía, nos pareció interesante mantener una mentalidad de abstracción y escalabilidad a la hora de plantear el proyecto. Esto hizo que tal vez se complejize mucho más, y por eso no llegamos a hacer todos los refactor y code smells necesarios para que queda más legible y performante. Por otro lado, nos desafió a buscar soluciones interesantes en C, donde encontramos un enriquecimiento en:

- División de responsabilidades
- Problemáticas por la falta de herencia
- Manejo de headers
- Directivas de preprocesamiento
- Manejo de ficheros y el posicionador vs manejo de strings
- Manejo de punteros a tipos de datos
- Manejo de punteros a funciones
- Tipo de dato void*
- makefile y problemáticas de compilación

Esto fue algo que sumó, por todos los topes con los que nos encontrábamos y porque no estabamos acostumbrados, ya que los dos programamos en lenguajes con GC y otras herramientas que facilitan muchas d e estas cosas.

¿Qué podemos agregar en una segunda iteración?

- Parser más genérico: Lo dejamos para el final subestimandolo, y nos enroscamos mucho en buscar expresiones regulares con sscanf/fscanf o un regex en C. No encontramos nada parecido y quedó bastante acoplado al html actual. Está en progreso en el branch **feature/parser_new** de github**.**
- Liberar memoria. Importantísimo.
- Limpiar algunos headers que tienen funciones de más o que deberían ir en otra entidad.
- Dividimos los headers entre funciones y types porque hay muchos que solo necesitaban incluir los tags y nos rompía por referencia cíclica. Luego descubrimos la precompilación y podríamos haber comprimido los headers en uno.
- Poner comentarios a todas las funciones.
- Configuraciones en archivos de texto para no tener que compilar de vuelta si se cambia el path de los files.
- Convención de nombres en snake_case. Los hicimos en camelCase y entendemos que en C sería ideal tenerlos en snake.

- Formatter. Quedo medio raro y poco entendible si un programador quiere agregar otro formateo.
- Manejo de arrays dinámicos.