Абстрактные типы данных

Модуль (1)

- Программу удобно рассматривать как набор независимых модулей.
- Модуль состоит из двух частей: интерфейса и реализации.
- *Интерфейс* описывает, что модуль делает. Он определяет идентификаторы, типы и подпрограммы, которые будут доступны коду, использующему этот модуль.
- *Реализация* описывает, как модуль выполняет то, что предлагает интерфейс.

Модуль (2)

- У модуля есть один интерфейс, но реализаций, удовлетворяющих этому интерфейсу, может быть несколько.
- Часть кода, которая использует модуль, называют *клиентом*.
- Клиент должен зависеть только от интерфейса, но не от деталей его реализации.

Преимущества использования модулей

- Абстракция (как средство борьбы со сложностью)

  Когда интерфейсы модулей согласованы, ответственность за реализацию каждого модуля делегируется определенному разработчику.

- Повторное использование

  Модуль может быть использован в другой программе.

- Сопровождение

  Можно заменить реализацию любого модуля, например, для улучшения производительности или переноса программы на другую платформу.

Модули в языке Си (1)

- В языке Си интерфейс описывается в заголовочном файле (*.h).
- В заголовочном файле описываются макросы, типы, переменные и функции, которые клиент может использовать.
- Клиент импортирует интерфейс с помощью директивы препроцессора include.

Модули в языке Си (2)

- Реализация интерфейса в языке Си представляется одним или несколькими файлами с расширением *.c.
- Реализация определяет переменные и функции, необходимые для обеспечения возможностей, описанных в интерфейсе.
- Реализация обязательно должна включать файл описания интерфейса, чтобы гарантировать согласованность интерфейса и реализации.

Типы модулей (1)

- Набор данных

  Набор связанных переменных и/или констант. В Си модули этого типа часто представляются только заголовочным файлом. (float.h, limits.h.)

- Библиотека

  Набор связанных функций.

- Абстрактный объект

  Набор функций, который обрабатывает скрытые данные.

Типы модулей (2)

- Абстрактный тип данных

  Абстрактный тип данных – это интерфейс, который определяет тип данных и операции над этим типом. Тип данных называется абстрактным, потому что интерфейс скрывает детали его представления и реализации.

Пример 1: абстрактный объект стек

stack_0.h

stack_0.c

main_0.c

Серьезный недостаток – не существует способа, создать несколько экземпляров стека.

Пример 2: «абстрактный» тип данных «стек»

stack_1.h

stack_1.c

main_1.c

К сожалению stack_t не является абстрактным типом данных, потому что stack_1.h показывает все детали реализации.

Неполный тип с языке Си (1)

- Стандарт Си описывает неполные типы как «типы которые описывают объект, но не предоставляют информацию нужную для определения его размера».

```
struct t;
```

- Пока тип неполный его использование ограничено.
- Описание неполного типа должно быть закончено где-то в программе.

Неполный тип с языке Си (2)

- Допустимо определять указатель на неполный тип

```
typedef struct t *T;
```

- Можно
  - определять переменные типа T;
  - передавать эти переменные как аргументы в функцию.
- Нельзя
  - применять операцию обращения к полю (->);
  - разыменовывать переменные типа T.

Пример 3: абстрактный тип данных «стек»

stack_2.h, stack_2.c, main_2.c

Стек реализован только для целых чисел.

Стек состоит максимум из 10 элементов.

Есть два решения для этой проблемы:

- использовать динамический массив;
- реализовать стек на базе линейного односвязного списка.

Пример 4: абстрактный тип данных «стек»

stack_4.h

stack_4.c

main_4.c

stack_5.h

stack_5.c

main_5.c

Трудности, улучшения и т.п. (1)

- Именование

  В примерах использовались имена функций, которые подходят для многих АТД (create, destroy, is_empty). Если в программе будет использоваться несколько разных АТД, это может привести к конфликту. Поэтому имеет смысл добавлять название АТД в название функций (stack_create, stack_destroy, stack_is_empty).

Трудности, улучшения и т.п. (2)

- Обработка ошибок
  - Интерфейс это своего рода контракт.
  - Интерфейс обычно описывает *проверяемые* ошибки времени выполнения и *непроверяемые* ошибки времени выполнения и исключения.
  - Реализация не гарантирует обнаружение непроверяемых ошибок времени выполнения. Хороший интерфейс избегает таких ошибок, но должен описать их.
  - Реализация гарантирует обнаружение проверяемых ошибок времени выполнение и информирование клиентского кода.

Трудности, улучшения и т.п. (3)

- «Общий» АТД
  - Хотелось бы чтобы стек мог «принимать» данные любого типа без модификации файла stack.h.
  - Программа не может создать два стека с данными разного типа.

  Решение – использовать void* как тип элемента, НО:
  - элементами могут быть динамически выделяемые объекты, но не данные базовых типов int, double;
  - стек может содержать указатели на что угодно, очень сложно гарантировать правильность.