Серыалызацыя структур данних в С

Вступ.

Визначення.

Теорыя серіалізації.

Порывняння різних реалізацій.

Алгоритм.

Приклади використання.

Структура данних — в мові програмування С, визначення списку змінних, які записуються під одним імям в блок памяті.

Порядок бійтів —

Обєкт —

Буфер —

Кделу

Сералязація -- це процес перетворення деякої структури в формат данних, наприклад бінарный, текстовый, XML, JSON та інші.

Десеріалізація це процес відновлення з серіалізованих данних початкової структури.

Серіалізація використовується для перетворення структури данних в формат який можна передавати по послідовному зєднанню(serial comunication), такому, як інтернет, або зберегти в файл. Наприклад, потрібно створити розподілений додаток, різні частини якого мають обмінюватися даними зі складною структурою. У такому випадку для типів даних, які передбачається передавати, пишеться код, який здійснює серіалізацію і десеріалізацію. Об'єкт заповнюється потрібними даними, потім викликається код серіалізації, в результаті виходить, XML-документ, або інший формат серіалізованих данних. Результат серіалізації передається приймаючій стороні по сокету. Додаток-одержувач створює об'єкт того ж типу і викликає код десеріалізації, в результаті отримується об'єкт з тими ж даними, що були в об'єкті програми-відправника. За такою схемою працює, серіалізація об'єктів через SOAP в Microsoft.NET.

Без серіалізації не обійтися в програмах які використовують інтернет для передачі складних структур данних.

В деяких мовах програмування як java, php, python, ocaml серіалізація є стандартною функцією. В стандарті С немає функцій для серіалізації, але їх можна реалізувати самостійно.

Найпростіший приклад серіалізації в С, це створення дампу памяті, наприклад:

```
struct some_struct s;
init_some_struct(&s);

FILE* f= fopen("./sdata", "w");
fwrite(&s, sizeof(s), 1, f)
fclose(f);
```

В результаті в файл запишеться копія блоку памяті структури s.
Для десералізації потрібно скопіювати данні з файлу в блок памяті структури.

В деяких ситуаціях такий підхід допустимий, але має недоліки

- якщо в структурі є вказівники, то в файл запишеться адреса вказівника, дані на які вказує вказівник не будуть записані.
- при десеріалізації цієї стректури на платформі з іншим порядком байт, структура міститиме данні відмінні від початкових.

Існує декілька бібліотек, що дозволяють серіалізувати данні в C, але жодна з них не підтримує структури, які мають однікові вказівники, це означає що за допомогою цих бібліотек не можна серіалізувати звязані списки(linked list) та дерева.

Для того, щоб на C можна було повноцінно сералізувати структури данних, була написана бібліотека cserialize.

Данна бібліотека може серіалізувати структури які мають вказівники, навіть якщо деякі вказівники дублюються, причому не важливо яку складність та вложеність мають структури.

Для використання цієї бібліотеки, для структури яку потрібно серіалізувати, потрібно визначити функцію-серіалізатор, це можна зробити за допомогою макросів бібліотеки.

Общий виглялад визначення серіалізатора

```
sfunc(<struct_name>, <modifer>)
  s<attr_type>(<attr>)
  s<attr2_type>(<attr2>)
funcs
```

<struct_name> -- імя структури.
<modifer> -- модифікатор, використовується, коли потрібно серіалізувати одну структуру різними способами.
<attr> -- імя атрибуту структури.
<attr_type> -- назва типу атрибуту.

sfunc() -- початок визначення функції.
Schar() -- код серіалізації для типу char.
sint16() -- код серіалізації для 16-ти бітного числа.
sint32() -- код серіалізації для 32-ти бітного числа.
sint64() -- код серіалізації для 64-ти бітного числа.
svoid_ptr(attr, size) — код серіалізації для блоку памяті на який вказує вказівник
schar_ptr() -- код серіалізації для null-terminated(строка останній символ якої \0) строки.
sstruct_ptr(attr) — код серіалізації для структури на яку вказує вказівник(attr), для цієї структури також потрібно визначити функцію-серіалізатор.
funcs() -- кінець визначення функції.

Під час компіляції замість макросів буде підставлено специфічна функція серіалізації для структури <struct_name>, яка матиме вигляд

```
struct buffer* <struct_name>_<modifer>_serializer(struct <struct_name>* s, ....) {
  ....
  //код серіалізації для данної структури
  ....
}
```

Визначення функції десеріалізатора має аналогічний виглад.

```
dsfunc(<struct_name>, <modifer>)
  ds<attr_type>(<attr>)
  ds<attr2_type>(<attr2>)
funcds
```

Макроси ds<attr_type>() повинні бути розміщенні в тому ж порядку, що й в функції-серіалізатора.

Наприклад є структура

```
struct book {
  const char* author;
```

```
  const char* name;
  int num_pages;
};
```

Визначення функції-серіалізатора буде виглядати так

```
sfunc(book)
  schar_ptr(author)
  schar_ptr(name)
  sint32(num_pages)
funcs
```

Десеріалізатора так

```
dsfunc(book)
  dschar_ptr(author)
  dschar_ptr(name)
  dsint32(num_pages)
funcds
```

Приклад використання

```
#include "serialize.h"

#include "sfuncs.h"//визначення серіалізатора та десеріалізатора

struct book {
  const char* author;
  const char* name;
  int num_pages;
};

int main() {
  struct book anarhy;
  anarhy.author = "Пётр Алексе́евич Кропо́ткин";
  anarhy.name = "Анархия";
  anarhy.num_pages = 512;

  struct buffer* book_buf = serialize(anarhy, book,);

  printf("str_buf->size = %i\n", str_buf->size);

  FILE* f = fopen("./sdata", "w");
  fwrite(str_buf->data, str_buf->size, 1, f);
  fclose(f);

  struct book* dbook = deserialize(str_buf->data, book,);

  printf("dbook.author = %s\n"
         "dbook.name = %s\n"
         "dbook.num_pages = %i\n",
          dbook->author, dbook->name, dbook->num_pages);
```

Ця програма виведе в термінал:

```
str_buf->size = 234;
dbook.author =  Пётр Алексе́евич Кропо́ткин
dbook.name = Анархия
dbook.num_pages = 512
```

Макрос serialize(<object>, <struct_name>, <modifer>) служить для виклику функції серіалізації, в данному випадку book__serialize().

Алгоритм серіалізації

Алгоритм роботи функції-серіалізатора

1. якщо це коренева функція(викликана макросом serialize), створюється буфер, в який будуть записуватисяя серіалізовані данні, інакше використовується переданий в аргумент буффер.
2. В масив ptr_offset записується вказівник на структуру, яку буде серіалізовувати ця функція.
3. Запис в буфер данних з кожного атрибуту структури
4. якщо якийсь атрибут є структурою, або вказівником на структуру, перевіряється чи був цей вказівник раніше серіалізований, якщо ні, викликається функція-серіалізатор для цієї структури, в яку передається вказівник на останнє використане місце в буфері. Викликана функція дописує в буфер данні з цього атрибуту.
5. Якщо це коренева функція, на початок буферу записується масив ptr_offset та кількість елементів масиву, в результат записується вказівник на буфер.
6. Вихід з функції

При серіалізації створюється буфері, в який записуються серіалізовані данні з структури, массив ptr_offset та кількість елементів масиву.

Загальний вигляд данних в буфері

|кількість серіалізованих вказівників|
|масив ptr_offset|
|серіалізована структура|

ptr_offset це масив, елеманти якого є парами (вказівник на серіалізовану структуру; відступ від початку буфера)