Библиотека Judy для работы с динамическими массивами

Компоненты библиотеки

Judy1 отображает индекс (**Index** - word) на бит.

JudyL отображает индекс (**Index** - word) на **Value** (word/указатель).

JudySL отображает индекс (строка с null-завершением) на **Value.**

JudyHS отображает индекс (массив байтов) размером **Length** на **Value.**

Judy представляет собой библиотеку C для создания и использования динамических массивов. Библиотека была предложена и реализована Doug Baskins из Hewlett-Packard.

Семейство функций и макросов поддерживает полностью динамические массивы, которые могут индексироваться 32- или 64-битовыми словами (в зависимости от размера word для процессора), строками с null-символом в конце или массивом байтов заданного размера. Динамический массив (разреженный) можно также считать функцией отображения или ассоциативной памятью.

Тип **Word_t** определен как *typedef unsigned long int в файле* **Judy.h** и должен иметь размер *sizeof(void *)*, т. е. указателя.

В функциях Judy1 **Index** имет тип **Word_t**, а **Value** - просто бит (**bit**) или флаг наличия (отсутствия **Index** в массиве. Этот можно рассматривать как огромное битовое поле (huge bitmap).

В функциях JudyL **Index** и **Value** имеют тип **Word_t**. В результате **JudyL** является чистым отображением слова на слово (указатель). **JudySL** и **JudyHL** базируются на этом свойстве **JudyL**.

В функциях JudySL **Index** является строкйо с null-символом в конце, а **Value** - **Word_t**.

В функциях JudyHS **Index** является массивом байтов размера **Length**, а **Value** имеет тип **Word_t**. Дополнение (май 2004 г.) в Judy является гибридом, в котором используются лучшие качества методов хещирования и Judy. Автор надеется, что **JudyHS** послужит хорошей заменой методов хэширования, когда размер хэш-таблицы меняется при росте численности. Корректно настроенный метод хеширования со статическим размером таблицы и численностью непобедим по скорости. Обнако **JudyHS** будет работать лучше с численностью, отличающейся от оптимального размера хэш-таблицы. **JudyHS** не теряет производительности в тех случаях, где это известно для алгоритмов хэширования. JudyHS не использует связный список для обработки конфликтов хэш-значений, применяя взамен дерево массивов **JudyL** и виртуальную хэш-таблицу размером 4 миллиарда.

Массивы Judy являются быстрыми и небольшими по памяти, не требуют найтройки для широкого диапазона типов набора индексов (последовательные, периодические, кластерные, случайные). Judy обычно превосходит по скорости и экономии памяти другие модели хранения данных, такие как skip-списки, связные списки, двоичные, троичные и b-деревья и даже хэширование, а также повышает эффективность работы при очень больших размерах данных.

Массив Judy создается просто путем определения пустого (null) указателя, а затем по этому указателю сохраняется (вставляется) первый элемент массива. Используемая массивом Judy память примерно пропорциональна численности (количеству элементов).

Judy имеет два интерфейса API - макросы C и вызовы функций. Поскольку макросы иногда быстрее и имеют более простой интерфейс обработки ошибок по сравнению с эквивалентными функциями, они являются предпочтительным вариантом.

Поскольку исходный (пустой) массив Judy представляется null-указателем, можно создать массив массивов Judy. Т. е. значениями массива Judy (**Value**), кроме Judy1, могут быть указатели на другие массивы Judy. Это существенно упрощает создание массивов произвольной размерности или значения **Index** (массивы JudySL и JudyHS используют для этого JudyL).

Файлы

/usr/share/doc/Judy/ - страницы руководства в формате HTML;
/usr/share/doc/Judy/demo/ - примеры файлов исходного кода.

Ошибки

На упрощение обработки ошибок в Judy было затрачено много сил и времени, что позволило сохранить гибкость и функциональность. Тема обработки ошибок достаточно скучна, но упускать ее нельзя. Из описанных здесь методов рекомендуется применять второй. Он создает наиболее быстрый код, используя минимальный объем памяти и требует написания дополнительного кода лишь для функций вставки и удаления. Метод совместим с двумя другими и предназначен для создания кода, который может обрабатывать отказы *malloc()* иначе, чем принято по умолчанию в Judy. Если принятый по умолчанию метод Judy работает хорошо, имеет смысл применять его.

Имеется две категории ошибок, возвращаемых Judy (и другими динамическими ADT):

1. Ошибки пользовательских программ, такие как повреждение памяти или непригодные указатели.
2. Нехватка памяти (отказ *malloc()*) при вставке (**I**nsert, **S**et) или удалении (**D**elete, **U**nset) для изменения массива Judy. Для вставки или удаления не всегда применяется функция *malloc()*, поэтому операции могут быть успешными даже при отказе *malloc()*.

Имеется три метода обработки ошибок при использовании макросов

1. **Принятый по умолчанию метод** выводит сообщение об ошибке в **stderr**. Например,

```
File 'YourCfile.c', line 1234: JudyLIns(), JU_ERRNO_* == 2, ID == 321
```

Это говорит об ошибке в строке 321 функции **JudyLIns()**. Проблема возникла в строке 1234 файла YourCfile.c, где произошел отказ **JLI()**. JU_ERRNO_* == 2 указывает JU_ERRNO_NOMEM (определена в файле **Judy.h**). Идентификатор ID указывает номер строки исходного файла, где произошла ошибка. Программа была прервана с возвратом *exit(1);*. По умолчанию обе категории ошибок Judy обрабатываются этим методом.

2. **Запрет обработки ошибок в макросах**

Когда в программе «нет ошибок», сообщения могут выдаваться лишь для отказов *malloc()* и все возвращаемые ошибки можно считать такими отказами. *Используя показанный ниже оператор* **#define**, можно отключить все проверки и вывод ошибок. К коду, который может приводить к отказам *malloc()*, потребуется добавить специальный код. Judy сохраняет в массиве данные, присутствовавшие до вызова, в случаях отказа *malloc()*[1].

```
#define JUDYERROR_NOTEST 1
```

в программном коде или

```
cc -DJUDYERROR_NOTEST sourcefile -lJudy
```

в командной строке.

```
//Пример использования метода 2 в программе.

JLI(PValue, PLArray, Index);
if (PValue == PJERR) goto out_of_memory_handling;
...

JLD(RC_int, PLArray, Index);
if (RC_int == JERR) goto out_of_memory_handling;
...

J1S(RC_int, P1Array, Index);
if (RC_int == JERR) goto out_of_memory_handling;
...

J1U(RC_int, P1Array, Index);
if (RC_int == JERR) goto out_of_memory_handling;
...
```

Отметим, что без определения JUDYERROR_NOTEST операция goto out_of_memory_handling не будет выполняться и компилятор исключит ее при оптимизации. Будет использован принятый по умолчанию метод использования макроса для вывода сообщения об ошибке, как описано выше. При определении JUDYERROR_NOTEST операция goto out_of_memory_handling будет выполняться в случае возникновения ошибки (отказ *malloc()*).

3. **Заданный пользователем макрос JUDYERROR()**

Макрос **JUDYERROR(), определенный в файле Judy.h** обеспечивает гибкую обработку ошибок, позволяющую программе использовать макросы массивов Judy вместо вызова функций. Можно использовать свой макрос **JUDYERROR()**, соответствующий реальным задачам. Ниже приведен пример возможного варианта взамен принятого по умолчанию. Он служит для различения двух типов ошибок и явно1 проверки остальных ошибок JU_ERRNO_NOMEM, которые могут возникать в программе.

```
// Пример Judy macro API для продолжения работы при нехватке памяти,
// вывода сообщения и возврата exit(1) при возникновении ошибки.

#ifndef JUDYERROR_NOTEST
#include <stdio.h>  // нужен для fprintf()

// Макрос, используемый Judy macro APIs для возврата кода -1:

#define JUDYERROR(CallerFile, CallerLine, JudyFunc, JudyErrno, JudyErrID) \
{                                                                         \
    if ((JudyErrno) != JU_ERRNO_NOMEM) /* не отказ malloc() */            \
    {                                                                     \
        (void) fprintf(stderr, "File '%s', line %d: %s(), "              \
            "JU_ERRNO_* == %d, ID == %d\n",                               \
            CallerFile, CallerLine,                                       \
            JudyFunc, JudyErrno, JudyErrID);                              \
        exit(1);                                                          \
    }                                                                     \
}
#endif // не определено JUDYERROR_NOTEST
```

Этот макрос обработки ошибок должен включаться в программу до оператора **#include <Judy.h>**.

Judy1

Макросы Judy1

Макросы Judy1 представляют собой библиотеку C для создания и использования динамического массива битов. Индексом массива может служить любое значение слова (word).

[1]В процессе тестирования Judy было найдено очень мало *malloc()*/ОС, где не возникало ошибок при отказах *malloc()*. Иногда приходилось ждать очень долго, поскольку в большинстве систем запускалась «бесконечная» подкачка.

Массив Judy1 эквивалентен массиву или отображению (map) битов, адресуемому **Index** (ключ). Массив может быть разреженным (sparse), а **Index** может быть любым значением (**Value**) размера word. Наличие индекса представляет установленный (set) бит, а такой бит представляет наличие индекса. Отсутствие индекса представляет сброшенный (unset) бит, а сброшенный бит представляет отсутствие индекса.

Массив Judy1 выделяется с указателем **NULL**

```
Pvoid_t PJ1Array = (Pvoid_t) NULL;
```

Память для поддержки массива выделяется при установке бита и освобождается при его сбросе. Если указатель Judy1 (**PJ1Array**) имеет значение NULL, это говорит, что все биты сброшены (для массива Judy1 не требуется памяти).

Как и обычные массивы Judy1, не имеет дубликатов индексов.

При использовании описанных здесь макросов вместо вызова функций Judy1 принятая по умолчанию обработка ошибок передает сообщение на стандартный вывод ошибок и завершает программу с кодом **exit(1)**. Другие методы обработки ошибок рассмотрены в разделе Ошибки.

Поскольку макросы иногда быстрее и включают более эффективную обработку ошибок по сравнению с функциями, их применение является предпочтительным в Judy1.

Синтаксис

```
cc [flags] sourcefiles -lJudy
#include <Judy.h>

int     Rc_int;                          // Код возврата - integer
Word_t  Rc_word;                         // Код возврата - unsigned word
Word_t  Index, Index1, Index2, Nth;

Pvoid_t PJ1Array = (Pvoid_t) NULL;       // Инициализация массива Judy1

J1S( Rc_int,  PJ1Array, Index);          // Judy1Set()
J1U( Rc_int,  PJ1Array, Index);          // Judy1Unset()
J1T( Rc_int,  PJ1Array, Index);          // Judy1Test()
J1C( Rc_word, PJ1Array, Index1, Index2); // Judy1Count()
J1BC(Rc_int,  PJ1Array, Nth, Index);     // Judy1ByCount()
J1FA(Rc_word, PJ1Array);                 // Judy1FreeArray()
J1MU(Rc_word, PJ1Array);                 // Judy1MemUsed()
J1F( Rc_int,  PJ1Array, Index);          // Judy1First()
J1N( Rc_int,  PJ1Array, Index);          // Judy1Next()
J1L( Rc_int,  PJ1Array, Index);          // Judy1Last()
J1P( Rc_int,  PJ1Array, Index);          // Judy1Prev()
J1FE(Rc_int,  PJ1Array, Index);          // Judy1FirstEmpty()
J1NE(Rc_int,  PJ1Array, Index);          // Judy1NextEmpty()
J1LE(Rc_int,  PJ1Array, Index);          // Judy1LastEmpty()
J1PE(Rc_int,  PJ1Array, Index);          // Judy1PrevEmpty()
```

Определения

J1S(Rc_int, PJ1Array, Index); // Judy1Set()

Устанавливает бит **Index** в массиве Judy1 **PJ1Array**. Возвращает **Rc_int** = 1 при сброшенном бите **Index** (нет ошибки) и 0 при установленном ранее бите (ошибка).

J1U(Rc_int, PJ1Array, Index); // Judy1Unset()

Сбрасывает бит **Index** в массиве Judy1 **PJ1Array**. Возвращает **Rc_int** = 1 при установленном ранее бите **Index** (нет ошибки) и 0 при сброшенном бите (ошибка).

J1T(Rc_int, PJ1Array, Index); // Judy1Test()

Проверяет установку бита **Index** в массиве Judy1 **PJ1Array**. Возвращает **Rc_int** = 1 при установленном бите **Index** (**Index** присутствует) и 0 при сброшенном бите (**Index** отсутствует).

J1C(Rc_word, PJ1Array, Index1, Index2); // Judy1Count()

Считает число индексов, присутствующих в массиве Judy1 **PJ1Array** между значениями **Index1** и **Index2** (включительно). Возвращает в **Rc_word** число интексов. Возврат **Value** = 0 может быть корректным для счетчика или указывать особый случай полностью заполненного массива (только на 32-битовых машинах). Более подробно это рассмотрено в описании функции Judy1Count(). Для подсчета числа присутствующих в массиве Judy1 индексов (численность) служит

```
J1C(Rc_word, PJ1Array, 0, -1);
```

Отметим, что -1 указывает максимальный индекс (все 1).

J1BC(Rc_int, PJ1Array, Nth, Index); // Judy1ByCount()

Находит **Nth**-й индекс, который присутствует в массиве Judy1 **PJ1Array** (**Nth** = 1 возвращает первый имеющийся индекс). Для указания последнего индекса в полностью заполненном массиве (присутствуют все индексы, что бывает редко) используется **Nth** = 0. Возвращает **Rc_int** = 1 и **Index** = **Nth**, если индекс найден, и **Rc_int** = 0 в противном случае (**Value** в **Index** не содержит полезной информации).

J1FA(Rc_word, PJ1Array); // Judy1FreeArray()

Освобождает весь массив Judy1 **PJ1Array** (быстрее цикла **J1N()**, **J1U()**). Возвращает в **Rc_word** число освобожденных битов и устанавливает **PJ1Array** = **NULL**.

J1MU(Rc_word, PJ1Array); // Judy1MemUsed()

Возвращает в **Rc_word** число байтов памяти, занятых массивом Judy1 **PJ1Array**. Это очень быстрый макрос и его можно применять после **J1S()** или **J1U()** практически без снижения производительности.

Макросы поиска Judy1

Средства поиска в Judy1 позволяют найти установленные или сброшенные биты в массиве. Поиск может быть включающим или исключающим и выполняется в обоих направлениях. Все средства поиска используют похожие

последовательности вызова. При успешном поиске в **Rc_int** возвращается 1 вместе с найденным индексом **Index**. При неудачном поиске **Rc_int** = 0, а **Index** не содержит полезной информации. Код возврата **Rc_int** должен проверяться до использования возвращенного **Index**, поскольку при поиске возможны отказы.

J1F(Rc_int, PJ1Array, Index); // Judy1First()
Включительный поиск первого присутствия индекса не меньше указанного **Index**. Для поиска первого элемента служит **Index** = 0. **J1F()** обычно служит для начала упорядоченного сканирования индексов в массиве Judy1.

J1N(Rc_int, PJ1Array, Index); // Judy1Next()
Исключительный поиск первого присутствия индекса больше указанного **Index**. **J1N()** обычно служит для продолжения упорядоченного сканирования индексов в массиве Judy1 или поиска «соседа» указанного индекса.

J1L(Rc_int, PJ1Array, Index); // Judy1Last()
Включительный поиск последнего присутствия индекса не больше указанного **Index**. Для поиска последнего индекса в массиве используется **Index** = -1 (все 1). **J1L()** обычно служит для начала упорядоченного сканирования в обратном направлении индексов в массиве Judy1.

J1P(Rc_int, PJ1Array, Index); // Judy1Prev()
Исключительный поиск предыдущего индекса меньше указанного **Index**. **J1P()** обычно служит для продолжения упорядоченного сканирования в обратном направлении индексов в массиве Judy1 или поиска «соседа» указанного индекса.

J1FE(Rc_int, PJ1Array, Index); // Judy1FirstEmpty()
Включительный поиск первого отсутствия индекса не меньше указанного **Index**. Для поиска первого отсутствующего элемента служит **Index** = 0.

J1NE(Rc_int, PJ1Array, Index); // Judy1NextEmpty()
Исключительный поиск первого отсутствия индекса больше указанного **Index**.

J1LE(Rc_int, PJ1Array, Index); // Judy1LastEmpty()
Включительный поиск последнего присутствия индекса не больше указанного **Index**. Для поиска последнего отсутствующего индекса в массиве используется **Index** = -1 (все 1)

J1PE(Rc_int, PJ1Array, Index); // Judy1PrevEmpty()
Исключительный поиск предыдущего индекса меньше указанного **Index**.

Пример

В приведенном ниже примере ошибки при вызове **J1S()** или **J1U()** вызывают пользовательскую процедуру process_malloc_failure. Этого не требуется при использовании принятого по умолчанию макроса **JUDYERROR()**, поскольку он завершает программу при любом отказе, включая ошибки *malloc()*.

```
#include <stdio.h>
#include <Judy.h>

int main()  {                    // Пример программы с Judy1 macro API
   Word_t Index;                 // индекс (или ключ)
   Word_t Rcount;                // счетчик индексов (установленных битов)
   Word_t Rc_word;               // возвращаемое значение (word)
   int    Rc_int;                // возвращаемое логическое значение (0 или 1)

   Pvoid_t PJ1Array = (Pvoid_t) NULL; // инициализация массива Judy1

   Index = 123456;
   J1S(Rc_int, J1Array, Index);  // Установка бита 123456
   if (Rc_int == JERR) goto process_malloc_failure;
   if (Rc_int == 1) printf("OK - bit successfully set at %lu\n", Index);
   if (Rc_int == 0) printf("BUG - bit already set at %lu\n", Index);

   Index = 654321;
   J1T(Rc_int, J1Array, Index);  // проверка установки бита 654321
   if (Rc_int == 1) printf("BUG - set bit at %lu\n", Index);
   if (Rc_int == 0) printf("OK - bit not set at %lu\n", Index);

   J1C(Rcount, J1Array, 0, -1);  // счетчик установленных битов массива
   printf("%lu bits set in Judy1 array\n", Rcount);

   Index = 0;
   J1F(Rc_int, J1Array, Index);  // найти первый бит, установленный в массиве
   if (Rc_int == 1) printf("OK - first bit set is at %lu\n", Index);
   if (Rc_int == 0) printf("BUG - no bits set in array\n");

   J1MU(Rc_word, J1Array);       // объем используемой памяти
   printf("%lu Indexes used %lu bytes of memory\n", Rcount, Rc_word);

   Index = 123456;
   J1U(Rc_int, J1Array, Index);  // сбросить бит 123456
   if (Rc_int == JERR) goto process_malloc_failure;
   if (Rc_int == 1) printf("OK - bit successfully unset at %lu\n", Index);
   if (Rc_int == 0) printf("BUG - bit was not set at %lu\n", Index);

   return(0);
}
```

Функции Judy1

Функции Judy1 обеспечивают библиотеку C для работы с динамическими массивами битов, в которых индексом служит любое значение типа word.

Для всех функций имеются эквивалентные макросы. Поскольку макросы иногда работают быстрее и обеспечивают более простой контроль ошибок, использование макросов предпочтительней вызова функций Judy1. Определения функций представлены ниже для полноты.

Одной из сложностей, связанных с вызовами функций Judy1, является решение вопроса о передаче указателя или его адреса. Поскольку функции, меняющие массив Judy1, должны менять и указатель на него, нужно передавать адрес указателя, а не сам указатель. Это ведет к трудно обнаруживаемым ошибкам в программах. На практике макросы позволяют компилятору обнаруживать программные ошибки при передаче указателей вместо адресов.

При вызове функций Judy1 используется дополнительный параметр по сравнению с соответствующими макросами. Этим параметром является указатель на структуру ошибки или **NULL** (если детали ошибок не возвращаются).

Ниже функции описаны в терминах их использования макросами (для случая **#define JUDYERROR_NOTEST 1**). Это является рекомендуемым использованием макросов после завершения отладки программ. Если макрос **JUDYERROR_NOTEST** не задан, объявляется структура ошибки для хранения информации, возвращаемой из функций Judy1 при возникновении ошибок.

Следует обращать внимание на размещение символа **&** в разных функциях.

Синтаксис

```
int    Judy1Set(PPvoid_t PPJ1Array, Word_t Index, PJError_t PJError);
int    Judy1Unset(PPvoid_t PPJ1Array, Word_t Index, PJError_t PJError);
int    Judy1Test(Pcvoid_t PJ1Array, Word_t Index, PJError_t PJError);
Word_t Judy1Count(Pcvoid_t PJ1Array, Word_t Index1, Word_t Index2, PJError_t PJError);
int    Judy1ByCount(Pcvoid_t PJ1Array, Word_t Nth, Word_t * PIndex, PJError_t PJError);
Word_t Judy1FreeArray(PPvoid_t PPJ1Array, PJError_t PJError);
Word_t Judy1MemUsed(Pcvoid_t PJ1Array);
int    Judy1First(Pcvoid_t PJ1Array, Word_t * PIndex, PJError_t PJError);
int    Judy1Next(Pcvoid_t PJ1Array, Word_t * PIndex, PJError_t PJError);
int    Judy1Last(Pcvoid_t PJ1Array, Word_t * PIndex, PJError_t PJError);
int    Judy1Prev(Pcvoid_t PJ1Array, Word_t * PIndex, PJError_t PJError);
int    Judy1FirstEmpty(Pcvoid_t PJ1Array, Word_t * PIndex, PJError_t PJError);
int    Judy1NextEmpty(Pcvoid_t PJ1Array, Word_t * PIndex, PJError_t PJError);
int    Judy1LastEmpty(Pcvoid_t PJ1Array, Word_t * PIndex, PJError_t PJError);
int    Judy1PrevEmpty(Pcvoid_t PJ1Array, Word_t * PIndex, PJError_t PJError);
```

Определения

Judy1Set(&PJ1Array, Index, &JError)

```
#define J1S(Rc_int, PJ1Array, Index) \
   Rc_int = Judy1Set(&PJ1Array, Index, PJE0)
```

Judy1Unset(&PJ1Array, Index, &JError)

```
#define J1U(Rc_int, PJ1Array, Index) \
   Rc_int = Judy1Unset(&PJ1Array, Index, PJE0)
```

Judy1Test(PJ1Array, Index, &JError)

```
#define J1T(Rc_int, PJ1Array, Index) \
   Rc_int = Judy1Test(PJ1Array, Index, PJE0)
```

Judy1Count(PJ1Array, Index1, Index2, &JError)

```
#define J1C(Rc_word, PJ1Array, Index1, Index2) \
   Rc_word = Judy1Count(PJ1Array, Index1, Index2, PJE0)
```

Возвращенное значение 0 может указывать ошибкой, быть пригодным значением счетчика или указывать особый случай полностью заполненного массива (только для 32-битовых машин). При необходимости для устранения неоднозначности можно использовать приведенный ниже код.

```
JError_t JError;
Rc_word = Judy1Count(PJ1Array, Index1, Index2, &JError);
if (Rc_word == 0)   {
    if (JU_ERRNO(&JError) == JU_ERRNO_NONE)
        printf("Judy1 array population == 0\n");
    if (JU_ERRNO(&JError) == JU_ERRNO_FULL)
        printf("Judy1 array population == 2^32\n");
    if (JU_ERRNO(&JError) == JU_ERRNO_NULLPPARRAY)
        goto NullArray;
    if (JU_ERRNO(&JError) >  JU_ERRNO_NFMAX)
        goto Null_or_CorruptArray;
}
```

Judy1ByCount(PJ1Array, Nth, &Index, &JError)

```
#define J1BC(Rc_int, PJ1Array, Nth, Index) \
   Rc_int = Judy1ByCount(PJ1Array, Nth, &Index, PJE0)
```

Judy1FreeArray(&PJ1Array, &JError)

```
#define J1FA(Rc_word, PJ1Array) \
   Rc_word = Judy1FreeArray(&PJ1Array, PJE0)
```

Judy1MemUsed(PJ1Array)

```
#define J1MU(Rc_word, PJ1Array) \
   Rc_word = Judy1MemUsed(PJ1Array)
```

Judy1First(PJ1Array, &Index, &JError)
```
#define J1F(Rc_int, PJ1Array, Index) \
   Rc_int = Judy1First(PJ1Array, &Index, PJE0)
```
Judy1Next(PJ1Array, &Index, &JError)
```
#define J1N(Rc_int, PJ1Array, Index) \
   Rc_int = Judy1Next(PJ1Array, &Index, PJE0)
```
Judy1Last(PJ1Array, &Index, &JError)
```
#define J1L(Rc_int, PJ1Array, Index) \
   Rc_int = Judy1Last(PJ1Array, &Index, PJE0)
```
Judy1Prev(PJ1Array, &Index, &JError)
```
#define J1P(Rc_int, PJ1Array, Index) \
   Rc_int = Judy1Prev(PJ1Array, &Index, PJE0)
```
Judy1FirstEmpty(PJ1Array, &Index, &JError)
```
#define J1FE(Rc_int, PJ1Array, Index) \
   Rc_int = Judy1FirstEmpty(PJ1Array, &Index, PJE0)
```
Judy1NextEmpty(PJ1Array, &Index, &JError)
```
#define J1NE(Rc_int, PJ1Array, Index) \
   Rc_int = Judy1NextEmpty(PJ1Array, &Index, PJE0)
```
Judy1LastEmpty(PJ1Array, &Index, &JError)
```
#define J1LE(Rc_int, PJ1Array, Index) \
   Rc_int = Judy1LastEmpty(PJ1Array, &Index, PJE0)
```
Judy1PrevEmpty(PJ1Array, &Index, &JError)
```
#define J1PE(Rc_int, PJ1Array, Index) \
   Rc_int = Judy1PrevEmpty(PJ1Array, &Index, PJE0)
```
Определения всех функций Judy, типов **Pvoid_t**, **Pcvoid_t**, **PPvoid_t**, **Word_t**, **JError_t** и **PJError_t**, констант **NULL**, **JU_ERRNO_***, **JERR** и **PJE0** приведены в файле **Judy.h** (/usr/include/Judy.h). Следует отметить, что вызывающие должны определять массивы Judy1 типа **Pvoid_t**, который можно передать функциям, принимающим **Pcvoid_t** (constant **Pvoid_t**), а также по адресу в функции, принимающие **PPvoid_t**.

JudyL

Макросы JudyL

Макросы JudyL представляют собой библиотеку C для создания и использования динамического массива слов (word). Индексом массива может служить любое значение слова (word).

Массив JudyL эквивалентен массиву значений **Value** типа word, адресуемому **Index** (ключ). Массив может быть разреженным (sparse), а **Index** может быть любым значением (**Value**) размера word. Память для поддержки массива выделяется при вставке пары «индекс-значение» и освобождается при ее удалении. Массив JudyL можно также считать отображением слова на другое слово (указатель).

Если указатель Judy1 (**PJ1Array**) имеет значение NULL, это говорит, что все биты сброшены (для массива Judy1 не требуется памяти).

Как и обычные массивы, JudyL не имеет дубликатов индексов.

Значение может применяться как скаляр, указатель на структуру или блок данных (включая другой массив Judy).

Массив JudyL выделяется с указателем **NULL**

```
Pvoid_t PJLArray = (Pvoid_t) NULL;
```
При использовании описанных здесь макросов вместо вызова функций JudyL принятая по умолчанию обработка ошибок передает сообщение на стандартный вывод ошибок и завершает программу с кодом **exit(1)**. Другие методы обработки ошибок рассмотрены в разделе Ошибки.

Поскольку макросы иногда быстрее и включают более эффективную обработку ошибок по сравнению с функциями, их применение является предпочтительным в JudyL.

Синтаксис
```
cc [flags] sourcefiles -lJudy

#include <Judy.h>

int     Rc_int;                         // Код возврата - integer
Word_t  Rc_word;                        // Код возврата - unsigned word
Word_t  Index, Index1, Index2, Nth;

PWord_t  PValue;                        // Указатель на возвращаемое значение
Pvoid_t PJLArray = (Pvoid_t) NULL;      // Инициализация массива JudyL

JLI( PValue,  PJLArray, Index);         // JudyLIns()
JLD( Rc_int,  PJLArray, Index);         // JudyLDel()
JLG( PValue,  PJLArray, Index);         // JudyLGet()
JLC( Rc_word, PJLArray, Index1, Index2); // JudyLCount()
JLBC(PValue,  PJLArray, Nth, Index);    // JudyLByCount()
JLFA(Rc_word, PJLArray);                // JudyLFreeArray()
JLMU(Rc_word, PJLArray);                // JudyLMemUsed()
JLF( PValue,  PJLArray, Index);         // JudyLFirst()
JLN( PValue,  PJLArray, Index);         // JudyLNext()
```

```
JLL( PValue,  PJLArray, Index);          // JudyLLast()
JLP( PValue,  PJLArray, Index);          // JudyLPrev()
JLFE(Rc_int,  PJLArray, Index);          // JudyLFirstEmpty()
JLNE(Rc_int,  PJLArray, Index);          // JudyLNextEmpty()
JLLE(Rc_int,  PJLArray, Index);          // JudyLLastEmpty()
JLPE(Rc_int,  PJLArray, Index);          // JudyLPrevEmpty()
```

JLI(PValue, PJLArray, Index) // JudyLIns()

Вставляет **Index** и **Value** в массив JudyL **PJLArray**. При успешной вставке устанавливается значение **Value** = 0. Если **Index** уже присутствует, **Value** не меняется. Возвращает указатель **PValue** на значение **Value**, который программа может использовать для чтения или изменения **Value**, пока не будет выполнен следующий макрос **JLI()** (вставка), **JLD()** (удаление) или **JLFA()** (очистка массива) для массива **PJLArray**. Например,

```
*PValue = 1234;
Value = *PValue;
```

Возвращает в **PValue** значение **PJERR**, если возникает отказ *malloc()*. Отметим, что макросы **JLI()** и **JLD()** реорганизуют массив JudyL, поэтому **PValue** из предыдущего вызова **JudyL** становится непригодным и должно быть обновлено.

JLD(Rc_int, PJLArray, Index) // JudyLDel()

Удаляет пару **Index**-**Value** из массива JudyL. Макрос возвращает **Rc_int** = 1 при успешном удалении и **Rc_int** = 0, если **Index** отсутствует. При отказе *malloc()* устанавливается **Rc_int** = **JERR**.

JLG(PValue, PJLArray, Index) // JudyLGet()

Возвращает указатель **PValue**, связанный с **Index** в массиве **PJLArray**. Макрос возвращает значение **Pvalue**, указывающее на **Value**. При отсутствии **Index** возвращается **PValue** = **NULL**, а пр отказе *malloc()* - **PValue** = **PJERR**.

JLC(Rc_word, PJLArray, Index1, Index2) // JudyLCount()

Возвращает число индексов, имеющихся в массиве JudyL **PJLArray** между значениями **Index1** и **Index2** (включительно). **Rc_word** содержит значение счетчика. Значение 0 может указывать отсутствие индексов. Для подсчета всех индексов в массиве JudyL используется вызов

```
JLC(Rc_word, PJLArray, 0, -1);
```

JLBC(PValue, PJLArray, Nth, Index) // JudyLByCount()

Находит **Nth**-й, присутствующий в массиве JudyL **PJLArray** (при **Nth** = 1 возвращается первый имеющийся индекс). Возвращает **PValue** с указанием на **Value** и **Index**, установленные для имеющегося **Nth**-ого индекса или **PValue** = **NULL**, если индекс отсутствует (значение **Index** в этом случае не имеет смысла).

JLFA(Rc_word, PJLArray) // JudyLFreeArray()

По указателю на массив JudyL полностью освобождает его гораздо быстрее чем в цикле **JLN()**, **JLD()**. Возвращает в **Rc_word** число освобожденных байтов и устанавливает **PJLArray** = **NULL**.

JLMU(Rc_word, PJLArray) // JudyLMemUsed()

Возвращает в **Rc_word** число байтов, выделенных *malloc()* для **PJLArray**. Это очень быстрый макрос и его можно использовать перед или после **JLI()** или **JLD()** практически без влияния на производительность.

Средства поиска JudyL

Макросы **JLF()**, **JLN()**, **JLL()**, **JLP()** позволяют выполнять поиск в массиве. Поиск может быть включительным или исключительным и выполняться в обоих направлениях. При успешном поиске **Index** содержит найденный индекс, а **PValue** - указатель на значение (**Value**) для найденного **Index**. Если поиск не дал результата, возвращается **PValue** = **NULL**, а **Index** не содержит полезной информации. Указатель **PValue** должен проверяться до использования **Index**, поскольку при поиске возможны отказы.

Макросы **JLFE()**, **JLNE()**, **JLLE()**, **JLPE()** позволяют искать отсутствующие (пустые) индексы в массиве. Поиск может быть включительным или исключительным и выполняется в обоих направлениях. При успешном поиске возвращается **Index** отсутствующего индекса и **Rc_int** = 1. При неудаче возвращается **Rc_int** = 0, а **Index** не содержит полезной информации. Значение **Rc_int** должно проверяться до использования **Index**, поскольку при поиске возможен отказ.

JLF(PValue, PJLArray, Index) // JudyLFirst()

Включительный поиск первого присутствующего индекса не меньше переданного **Index**. При **Index** = 0 отыскивается первый имеющийся индекс. **JLF()** обычно применяется для начала упорядоченного сканирования имеющихся в JudyL индексов.

JLN(PValue, PJLArray, Index) // JudyLNext()

Исключительный поиск первого присутствующего индекса больше переданного **Index**. **JLN()** обычно служит для продолжения упорядоченного сканирования индекса JudyL или поиска «соседа» данного индекса.

JLL(PValue, PJLArray, Index) // JudyLLast()

Включительный поиск последнего присутствующего индекса не больше переданного **Index**. При **Index** = -1 (все 1) отыскивается последний индекс массива. **JLL()** обычно применяется для упорядоченного сканирования присутствующих в массиве JudyL индексов в обратном направлении.

JLP(PValue, PJLArray, Index) // JudyLPrev()

Исключительный поиск предыдущего индекса меньше переданного **Index**. **JLP()** обычно служит для продолжения упорядоченного сканирования присутствующих в массиве JudyL индексов в обратном направлении.

JLFE(Rc_int, PJLArray, Index) // JudyLFirstEmpty()

Включительный поиск первого отсутствующего индекса не меньше переданного **Index**. **Index** = 0 задает поиск первого отсутствующего в массиве индекса.

JLNE(Rc_int, PJLArray, Index) // JudyLNextEmpty()

Исключительный поиск следующего отсутствующего индекса больше переданного **Index**.

JLLE(Rc_int, PJLArray, Index) // JudyLLastEmpty()

Включительный поиск последнеого отсутствующего индекса не больше переданного **Index**. При **Index** = -1 (все 1) отыскивается последний индекс, отсутствующий в массиве.

JLPE(Rc_int, PJLArray, Index) // JudyLPrevEmpty()

Исключительный поиск предыдущего индекса меньше переданного **Index**.

Многомерные массивы JudyL

Запись указателя на другой массив JudyL в **Value** массива JudyL обеспечивает простой способ поддержки динамических многомерных массивов. Такие массивы (деревья), созданные с помощью JudyL, являются очень

быстрыми и нетребовательными к памяти (фактически, так реализованы JudySL и JudyHS). Таким способом можно реализовать произвольную размерность. Для завершения числа измерений (дерева), указатель в **Value** помечается как не указывающий на другой массив Judy с помощью флага **JLAP_INVALID** в младших битах указателя[1]. После удаления флага **JLAP_INVALID** значение используется как указатель на пользовательские данные. Флаг **JLAP_INVALID** определен в файле **Judy.h**.

Ниже приведен фрагмент кода, который можно использовать для проверки указания на другой массив JudyL.

```
PValue = (PWord_t)PMultiDimArray;

for (Dim = 0; ;Dim++)      {
   if (PValue == (PWord_t)NULL) goto IndexNotFound;

   /* Переход к следующему измерению в массиве. */
   JLG(PValue, (Pvoid_t)*PValue, Index[Dim]);

   /* Проверка указателя на пользовательский буфер. */
   if (*PValue & JLAP_INVALID)) break;
}
UPointer = (UPointer_t) (*PValue & ~JLAP_INVALID);  // mask and cast.
printf("User object pointer is 0x%lx\n", (Word_t) UPointer);
      ...
```

Этот код работает, поскольку *malloc()* гарантирует возврат указателя с младшими битами, имеющими значение 0x0. Флаг **JLAP_INVALID** должен удаляться перед использованием указателя.

Пример

Считывание последовательности пар «индекс-значение» со стандартного ввода, сохранение в массиве и вывод в отсортированном виде.

```
#include <stdio.h>
#include <Judy.h>

Word_t   Index;                    // индекс массива
Word_t   Value;                    // значение элемента массива
Word_t * PValue;                   // указатель на значение элемента массива
int      Rc_int;                   // код возврата

Pvoid_t  PJLArray = (Pvoid_t) NULL; // Инициализация массива JudyL

while (scanf("%lu %lu", &Index, &Value))      {
    JLI(PValue, PJLArray, Index);
    If (PValue == PJERR) goto process_malloc_failure;
    *PValue = Value;               // запись нового значения
}
// Затем перебираются все сохраненные индексы в порядке сортировки сначала по возрастанию,
// потом по убыванию и во втором проходе каждый индекс удаляется.

Index = 0;
JLF(PValue, PJLArray, Index);
while (PValue != NULL)     {
    printf("%lu %lu\n", Index, *PValue));
    JLN(PValue, PJLArray, Index);
}

Index = -1;
JLL(PValue, PJLArray, Index);
while (PValue != NULL)     {
    printf("%lu %lu\n", Index, *PValue));

    JLD(Rc_int, PJLArray, Index);
    if (Rc_int == JERR) goto process_malloc_failure;

    JLP(PValue, PJLArray, Index);
}
```

Функции JudyL

Функции JudyL обеспечивают библиотеку C для создания и работы с динамическими массивами слов (word), использующими в качестве индекса любые значения слов (word).

Для каждой функции имеется эквивалентный макрос. Поскольку макросы иногда быстрее и проще в обработке ошибок, они являются предпочтительными. Определения функций приведены здесь лишь для полноты.

Одной из сложностей, связанных с вызовами функций JudyL, является решение вопроса о передаче указателя или его адреса. Поскольку функции, меняющие массив JudyL, должны менять и указатель на него, нужно передавать адрес указателя, а не сам указатель. Это ведет к трудно обнаруживаемым ошибкам в программах. На практике макросы позволяют компилятору обнаруживать программные ошибки при передаче указателей вместо адресов.

[1]В текущей версии **Judy.h** значение этого флага 0x4 было заменено на 0x1, чтобы можно было использовать функцию *malloc(),* не выравнивающую выделенную память по 8-битовой границе (например, старая версия valgrind).

При вызове функций JudyL используется дополнительный параметр по сравнению с соответствующими макросами. Этим параметром является указатель на структуру ошибки или **NULL** (если детали ошибок не возвращаются).

Ниже функции описаны в терминах их использования макросами (для случая **#define JUDYERROR_NOTEST 1**). Это является рекомендуемым использованием макросов после завершения отладки программ. Если макрос **JUDYERROR_NOTEST** не задан, объявляется структура ошибки для хранения информации, возвращаемой из функций JudyL при возникновении ошибок.

Следует обращать внимание на размещение символа **&** в разных функциях.

Синтаксис

```
PPvoid_t JudyLIns(PPvoid_t PPJLArray, Word_t    Index, PJError_t PJError);
int      JudyLDel(PPvoid_t PPJLArray, Word_t    Index, PJError_t PJError);
PPvoid_t JudyLGet(Pcvoid_t  PJLArray, Word_t    Index, PJError_t PJError);
Word_t   JudyLCount(Pcvoid_t PJLArray, Word_t Index1, Word_t Index2, PJError_t PJError);
PPvoid_t JudyLByCount(Pcvoid_t PJLArray, Word_t Nth, Word_t * PIndex, PJError_t PJError);
Word_t   JudyLFreeArray(PPvoid_t PPJLArray, PJError_t PJError);
Word_t   JudyLMemUsed(Pcvoid_t  PJLArray);
PPvoid_t JudyLFirst(Pcvoid_t  PJLArray, Word_t * PIndex, PJError_t PJError);
PPvoid_t JudyLNext(Pcvoid_t  PJLArray, Word_t * PIndex, PJError_t PJError);
PPvoid_t JudyLLast(Pcvoid_t  PJLArray, Word_t * PIndex, PJError_t PJError);
PPvoid_t JudyLPrev(Pcvoid_t  PJLArray, Word_t * PIndex, PJError_t PJError);
int      JudyLFirstEmpty(Pcvoid_t  PJLArray, Word_t * PIndex, PJError_t PJError);
int      JudyLNextEmpty(Pcvoid_t PJLArray, Word_t * PIndex, PJError_t PJError);
int      JudyLLastEmpty(Pcvoid_t PJLArray, Word_t * PIndex, PJError_t PJError);
int      JudyLPrevEmpty(Pcvoid_t PJLArray, Word_t * PIndex, PJError_t PJError);
```

Определения

JudyLIns(&PJLArray, Index, &JError)

```
#define JLI(PValue, PJLArray, Index)  \
   PValue = JudyLIns(&PJLArray, Index, PJE0)
```

JudyLDel(&PJLArray, Index, &JError)

```
#define JLD(Rc_int, PJLArray, Index)  \
   Rc_int = JudyLDel(&PJLArray, Index, PJE0)
```

JudyLGet(PJLArray, Index, &JError)

```
#define JLG(PValue, PJLArray, Index)  \
   PValue = JudyLGet(PJLArray, Index, PJE0)
```

JudyLCount(PJLArray, Index1, Index2, &JError)

```
#define JLC(Rc_word, PJLArray, Index1, Index2)  \
   Rc_word = JudyLCount(PJLArray, Index1, Index2, PJE0)
```

JudyLByCount(PJLArray, Nth, &Index, &JError)

```
#define JLBC(PValue, PJLArray, Nth, Index) \
   PValue = JudyLByCount(PJLArray, Nth, &Index, PJE0)
```

JudyLFreeArray(&PJLArray, &JError)

```
#define JLFA(Rc_word, PJLArray) \
   Rc_word = JudyLFreeArray(&PJLArray, PJE0)
```

JudyLMemUsed(PJLArray)

```
#define JLMU(Rc_word, PJLArray) \
   Rc_word = JudyLMemUsed(PJLArray)
```

JudyLFirst(PJLArray, &Index, &JError)

```
#define JLF(PValue, PJLArray, Index) \
   PValue = JudyLFirst(PJLArray, &Index, PJEO)
```

JudyLNext(PJLArray, &Index, &JError)

```
#define JLN(PValue, PJLArray, Index) \
   PValue = JudyLNext(PJLArray, &Index, PJEO)
```

JudyLLast(PJLArray, &Index, &JError)

```
#define JLL(PValue, PJLArray, Index) \
   PValue = JudyLLast(PJLArray, &Index, PJEO)
```

JudyLPrev(PJLArray, &Index, &JError)

```
#define JLP(PValue, PJLArray, Index) \
   PValue = JudyLPrev(PJLArray, &Index, PJEO)
```

JudyLFirstEmpty(PJLArray, &Index, &JError)

```
#define JLFE(Rc_int, PJLArray, Index) \
   Rc_int = JudyLFirstEmpty(PJLArray, &Index, PJEO)
```

JudyLNextEmpty(PJLArray, &Index, &JError)

```
#define JLNE(Rc_int, PJLArray, Index) \
   Rc_int = JudyLNextEmpty(PJLArray, &Index, PJEO)
```

JudyLLastEmpty(PJLArray, &Index, &JError)

```
#define JLLE(Rc_int, PJLArray, Index) \
   Rc_int = JudyLLastEmpty(PJLArray, &Index, PJEO)
```

JudyLPrevEmpty(PJLArray, &Index, &JError)

```
#define JLPE(Rc_int, PJLArray, Index) \
   Rc_int = JudyLPrevEmpty(PJLArray, &Index, PJEO)
```

Определения всех функций Judy, типов **Pvoid_t**, **Pcvoid_t**, **PPvoid_t**, **Word_t**, **JError_t** и **PJError_t**, констант **NULL**, **JU_ERRNO_***, **JERR** и **PJE0** приведены в файле **Judy.h** (/usr/include/Judy.h). Следует отметить, что вызывающие должны определять массивы Judy1 типа **Pvoid_t**, который можно передать функциям, принимающим **Pcvoid_t** (constant **Pvoid_t**), а также по адресу в функции, принимающие **PPvoid_t**.

Большинство функций **JudyL** возвращает **PPvoid_t**, поскольку значения в массиве могут быть ссылками на другие объекты, или **Word_t ***, когда нужен указатель на **Value**, а не указатель на указатель.

JudySL

Макросы JudySL

Макросы JudySL обеспечивают библиотеку C для создания и работы с динамическими массивами, использующими строки с null-символом в конце в качестве **Index** (ассоциативный массив).

Массив JudySL эквивалентен отсортированному набору строк, с каждой из которых связано значение **Value** типа word. **Value** адресуются индексами **Index** (ключ), которые представляют собой строки произвольного размера, завершающиеся null-символом. Память для массива выделяется при вставке пар «индекс-значение» и и освобождается при удалении пар. Это форма ассоциативного массива, где элементы сортируются лексически (с учетом регистра) по индексам. Это можно рассматривать как

```
void * JudySLArray["Toto, I don't think we're in Kansas any more"];
```

Массив JudySL создается с указателем **NULL**

```
Pvoid_t PJSLArray = (Pvoid_t) NULL;
```

Как в обычных массивах индексы не дублируются.

При использовании описанных здесь макросов вместо вызова функций JudySL принятая по умолчанию обработка ошибок выводит сообщения на стандартный вывод ошибок и завершает программу с **exit(1)**.

Синтаксис

```
cc [flags] sourcefiles -lJudy

#include <Judy.h>

#define MAXLINELEN 1000000           // задает максимальный размер строк

Word_t * PValue;                     // элемент массива JudySL
uint8_t  Index[MAXLINELEN];          // строка
int      Rc_int;                     // возвращаемое значение
Word_t   Rc_word;                    // полное слово возвращаемого значения

Pvoid_t PJSLArray = (Pvoid_t) NULL;  // Инициализация массива JudySL

JSLI(PValue,  PJSLArray, Index);     // JudySLIns()
JSLD(Rc_int,  PJSLArray, Index);     // JudySLDel()
JSLG(PValue,  PJSLArray, Index);     // JudySLGet()
JSLFA(Rc_word, PJSLArray);           // JudySLFreeArray()
JSLF(PValue,  PJSLArray, Index);     // JudySLFirst()
JSLN(PValue,  PJSLArray, Index);     // JudySLNext()
JSLL(PValue,  PJSLArray, Index);     // JudySLLast()
JSLP(PValue,  PJSLArray, Index);     // JudySLPrev()
```

JSLI(PValue, PJSLArray, Index) // JudySLIns()

Вставляет строку **Index** и **Value** в массив JudySL **PJSLArray**. При успешной вставке **Index** устанавливается **Value** = 0. Если **Index** уже присутствует, **Value** не меняется. Макрос возвращает указатель **PValue** на значение**Value** индекса **Index**. Программы должны использовать этот указатель для изменения **Value**, например,

```
*PValue = 1234;
```

Макросы **JSLI()** и **JSLD** реорганизуют массив JudySL, поэтому возвращенные предыдущими вызовами **JudySL** указатели теряют силу и должны быть определены заново.

JSLD(Rc_int, PJSLArray, Index) // JudySLDel()

Удаляет заданную пару **Index**-**Value** (элемент) из массива JudySL. Возвращает **Rc_int** = 1 при успешном удалении и **Rc_int** = 0, если **Index** отсутствует в массиве.

JSLG(PValue, PJSLArray, Index) // JudySLGet()

Возвращает указатель на **Value** индекса **Index** или **PValue** = **NULL**, если **Index** отсутствует в массиве.

JSLFA(Rc_word, PJSLArray) // JudySLFreeArray()

По указателю на массив JudySL (**PJSLArray**) освобождает массив полностью (гораздо быстрей, нежели цикл **JSLN()**, **JSLD()**). Возвращает в **Rc_word** число освобожденных байтов и **PJSLArray** = NULL.

Средства поиска JudySL

Средства поиска JudySL обеспечивают поиск индексов в массиве (включительный или исключительный) в прямом и обратном направлении.

При успешном поиске **Index** содержит найденный индекс, а **PValue** - указатель на значение (**Value**) для найденного **Index**. Если поиск не дал результата, возвращается **PValue** = **NULL**, а **Index** не содержит полезной информации. Указатель **PValue** должен проверяться до использования **Index**, поскольку при поиске возможны отказы. Для учета всех возможных результатов буфер **Index** должен иметь размер не меньше максимально длинной строки массива.

JSLF(PValue, PJSLArray, Index) // JudySLFirst()

Включительный поиск первого имеющегося индекса не меньше переданной строки **Index**. Поиск с пустой (null) строкой индекса будет возвращать первый индекс в массиве. **JSLF()** обычно применяется для начала упорядоченного сканирования действительных индексов массива JudySL.

```
uint8_t Index[MAXLINELEN];
strcpy (Index, "");
JSLF(PValue, PJSLArray, Index);
```

JSLN(PValue, PJSLArray, Index) // JudySLNext()

Исключительный поиск следующего индекса больше переданной строки **Index**. **JSLN()** обычно служит для продолжения упорядоченного сканирования действительных индексов в массиве JudySL или поиска «соседа» заданного индекса.

JSLL(PValue, PJSLArray, Index) // JudySLLast()

Включительный поиск последнего имеющегося индекса не больше переданной строки **Index**. Начало поиска со строки с максимальным значением (например, самой длинной строки с 0xff байтов) возвратит последний индекс в массиве. **JSLL()** обычно служит для начала упорядоченного сканирования действительных индексов массива JudySL в обратном направлении.

JSLP(PValue, PJSLArray, Index) // JudySLPrev()

Исключительный поиск предыдущего индекса меньше переданной строки **Index**. **JSLP()** обычно служит для продолжения упорядоченного сканирования действительных индексов массива JudySL в обратном направлении или поиска «соседа» указанного индекса.

Пример программы сортировки строк

```
#include <stdio.h>
#include <Judy.h>

#define MAXLINE 1000000                 // максимальный размер строки (длина)

uint8_t   Index[MAXLINE];               // строка для вставки

int     // Синтаксис:  JudySort < file_to_sort
main()
{
    Pvoid_t   PJArray = (PWord_t)NULL;  // Массив Judy.
    PWord_t   PValue;                   // Элемент массива Judy.
    Word_t    Bytes;                    // Размер массива JudySL.

    while (fgets(Index, MAXLINE, stdin) != (char *)NULL)
    {
        JSLI(PValue, PJArray, Index);   // Сохранить строку в массиве.
        if (PValue == PJERR)            // Недостаточно памяти?
        {                               // Обработка нехватки памяти.
            printf("Malloc failed -- get more ram\n");
            exit(1);
        }
        ++(*PValue);                    // Число экземпляров строк.
    }
    Index[0] = '\0';                    // Начать с наименьшей строки.
    JSLF(PValue, PJArray, Index);       // Находит первую строку.
    while (PValue != NULL)
    {
        while ((*PValue)--)             // Выводит дубликаты.
            printf("%s", Index);
        JSLN(PValue, PJArray, Index);   // находит следующую строку.
    }
    JSLFA(Bytes, PJArray);              // Освобождает массив.

    fprintf(stderr, "The JudySL array used %lu bytes of memory\n", Bytes);
    return (0);
}
```

Функции JudySL

Функции JudySL обеспечивают библиотеку C для создания и работы с динамическими массивами строк, завершающихся null-символом (ассоциативный массив).

Для каждой функции имеется эквивалентный макрос. Поскольку макросы иногда быстрее и проще в обработке ошибок, они являются предпочтительными. Определения функций приведены здесь лишь для полноты.

Одной из сложностей, связанных с вызовами функций JudySL, является решение вопроса о передаче указателя или его адреса. Поскольку функции, меняющие массив JudySL, должны менять и указатель на него, нужно передавать адрес указателя, а не сам указатель. Это ведет к трудно обнаруживаемым ошибкам в программах. На практике макросы позволяют компилятору обнаруживать программные ошибки при передаче указателей вместо адресов.

При вызове функций JudySL используется дополнительный параметр по сравнению с соответствующими макросами. Этим параметром является указатель на структуру ошибки или **NULL** (если детали ошибок не возвращаются).

Ниже функции описаны в терминах их использования макросами (для случая **#define JUDYERROR_NOTEST 1**). Это является рекомендуемым использованием макросов после завершения отладки программ. Если макрос **JUDYERROR_NOTEST** не задан, объявляется структура ошибки для хранения информации, возвращаемой из функций JudySL при возникновении ошибок.

Следует обращать внимание на размещение символа **&** в разных функциях.

Синтаксис

```
PPvoid_t JudySLIns(PPvoid_t PPJSLArray, const uint8_t * Index, PJError_t PJError);
int      JudySLDel(PPvoid_t PPJSLArray, const uint8_t * Index, PJError_t PJError);
```

```
PPvoid_t JudySLGet(Pcvoid_t PJSLArray, const uint8_t * Index, PJError_t PJError);
Word_t   JudySLFreeArray(PPvoid_t PPJSLArray, PJError_t PJError);
PPvoid_t JudySLFirst(Pcvoid_t PJSLArray, uint8_t * Index, PJError_t PJError);
PPvoid_t JudySLNext(Pcvoid_t PJSLArray, uint8_t * Index, PJError_t PJError);
PPvoid_t JudySLLast(Pcvoid_t PJSLArray, uint8_t * Index, PJError_t PJError);
PPvoid_t JudySLPrev(Pcvoid_t PJSLArray, uint8_t * Index, PJError_t PJError);
```

Определения

JudySLIns(&PJSLArray, Index, &JError)
```
#define JSLI(PValue, PJSLArray, Index) \
   PValue = JudyLIns(&PJSLArray, Index, PJE0)
```
JudySLDel(&PJSLArray, Index, &JError)
```
#define JSLD(Rc_int, PJSLArray, Index) \
   Rc_int = JudySLDel(&PJSLArray, Index, PJE0)
```
JudySLGet(PJSLArray, Index, &JError)
```
#define JSLG(PValue, PJSLArray, Index) \
   PValue = JudySLIns(PJSLArray, Index, PJE0)
```
JudySLFreeArray(&PJSLArray, &JError)
```
#define JSLFA(Rc_word, PJSLArray) \
   Rc_word = JudySLFreeArray(&PJSLArray, PJE0)
```
JudySLFirst(PJSLArray, Index, &JError)
```
#define JSLF(PValue, PJSLArray, Index) \
   PValue = JudySLFirst(PJSLArray, Index, PJE0)
```
JudySLNext(PJSLArray, Index, &JError)
```
#define JSLN(PValue, PJSLArray, Index) \
   PValue = JudySLNext(PJSLArray, Index, PJE0)
```
JudySLLast(PJSLArray, Index, &JError)
```
#define JSLL(PValue, PJSLArray, Index) \
   PValue = JudySLLast(PJSLArray, Index, PJE0)
```
JudySLPrev(PJSLArray, Index, &JError)
```
#define JSLP(PValue, PJSLArray, Index) \
   PValue = JudySLPrev(PJSLArray, Index, PJE0)
```

Определения всех функций Judy, типов **Pvoid_t**, **Pcvoid_t**, **PPvoid_t**, **Word_t**, **JError_t** и **PJError_t**, констант **NULL**, **JU_ERRNO_***, **JERR** и **PJE0** приведены в файле **Judy.h** (/usr/include/Judy.h). Следует отметить, что вызывающие должны определять массивы Judy1 типа **Pvoid_t**, который можно передать функциям, принимающим **Pcvoid_t** (constant **Pvoid_t**), а также по адресу в функции, принимающие **PPvoid_t**.

Большинство функций **JudySL** возвращает **PPvoid_t**, поскольку значения в массиве могут быть ссылками на другие объекты, или **Word_t ***, когда нужен указатель на **Value**, а не указатель на указатель.

JudyHS

Макросы JudyHS

Библиотека C для создания и работы с динамическими массивами, использующими массив байтов размера **Length** в качестве **Index** и word как **Value**.

Массив JudyHS эквивалентен массиву значений или указателей размера word. Index служит указателем на массив байтов заданного размера Length. Вместо использования строк с null-завершением (как в JudySL) здесь могут применяться строки с любыми битами, включая null. Это дополнение (май 2004 г.) к массивам Judy является комбинацией методов хэширования и Judy. В JudyHS не возникает возможности снижения производительности за счет знания алгоритма хэширования.

Поскольку JudyHS базируется на хэшировании, индексы не сохраняются в каком-либо определенном порядке. Поэтому функции JudyHSFirst(), JudyHSNext(), JudyHSPrev() и JudyHSLast() для поиска соседей не имеют практического значения. Значение Length для каждого массива байтов может меняться от 0 до предела malloc() (около 2 Гбайт).

Отличием JudyHS является скорость и расширяемость при сохранении эффективности использования памяти. Скорость сопоставима с лучшими методами хэширования, а эффективность использования памяти близка к связным спискам для таких же Index и Value. Массивы JudyHS рассчитаны на работу с числом индексов от 0 до миллиардов.

Массив JudyHS создается с указателем NULL.

```
Pvoid_t PJHSArray = (Pvoid_t) NULL;
```

Поскольку макросы проще и эффективней в части обработки ошибок, они являются предпочтительными по сравнению с функциями JudyHS.

Синтаксис

```
cc [flags] sourcefiles -lJudy

#include <Judy.h>

Word_t  * PValue;                          // элемент массива JudyHS
int       Rc_int;                          // код возврата
Word_t    Rc_word;                         // полное значение возвращаемого слова
Pvoid_t   PJHSArray = (Pvoid_t) NULL;      // инициализация массива JudyHS
uint8_t * Index;                           // указатель на массив байтов
Word_t    Length;                          // число байтов в Index
```

```
JHSI( PValue,  PJHSArray, Index, Length);   // JudyHSIns()
JHSD( Rc_int,  PJHSArray, Index, Length);   // JudyHSDel()
JHSG( PValue,  PJHSArray, Index, Length);   // JudyHSGet()
JHSFA(Rc_word, PJHSArray);                  // JudyHSFreeArray()
```

Определения

JHSI(PValue, PJHSArray, Index, Length) // JudyHSIns()

По указателю на массив JudyHS (**PJHSArray**) вставляет строку **Index** размера **Length** и **Value** в массив JudyHS **PJHSArray**. При успешной вставке **Index** устанавливается **Value** = 0. Если **Index** уже есть в массиве, **Value** не меняется. Возвращаемое значение **PValue** указывает на **Value**. Программам следует использовать этот указатель для изменения **Value**, например,

```
Value = *PValue;
*PValue = 1234;
```

Отметим, что **JHSI()** и **JHSD** могут реорганизовать массив JudyHS, поэтому возвращенные из предыдущих вызовов **JudyHS** указатели должны быть обновлены (с помощью **JHSG()**).

JHSD(Rc_int, PJHSArray, Index, Length) // JudyHSDel()

По указателю на массив JudyHS (**PJHSArray**) удаляет указанный **Index** вместе с **Value** из массива JudyHS. Возвращает **Rc_int** = 1 при успешном удалении и **Rc_int** = 0, если **Index** отсутствует.

JHSG(PValue, PJHSArray, Index, Length) // JudyHSGet()

По указателю на массив JudyHS (**PJHSArray**) находит значение **Value**, связанное с **Index**. Возвращает указатель **PValue** на значение **Value** для **Index** или **PValue** = **NULL** при отсутствии **Index**.

JHSFA(Rc_word, PJHSArray) // JudyHSFreeArray()

По указателю на массив JudyHS (**PJHSArray**) освобождает массив целиком. Возвращает в **Rc_word** число освобожденных байтов и **PJHSArray** = NULL.

Пример

Приведенный ниже фрагмент кода будет выводить дубликаты строк с их номерами при вводе с *stdin*.

```
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <Judy.h>

// Команда компиляции cc -O PrintDupLines.c -lJudy -o PrintDupLines

#define MAXLINE 1000000                 /* Максимальный размер строки */
uint8_t   Index[MAXLINE];               // Проверяемая строка

int     // Команда PrintDupLines < file
main() {
    Pvoid_t   PJArray = (PWord_t)NULL;  // Массив Judy.
    PWord_t   PValue;                   // Указатель на элемент массива Judy.
    Word_t    Bytes;                    // Размер массива JudyHS.
    Word_t    LineNumb = 0;             // Номер текущей строки.
    Word_t    Dups = 0;                 // Число строк-дубликатов.

    while (fgets(Index, MAXLINE, stdin) != (char *)NULL)    {
        LineNumb++;                     // Номер строки.

        // Сохранение строки в массиве
        JHSI(PValue, PJArray, Index, strlen(Index));
        if (PValue == PJERR)     {      // См. Ошибки.
            fprintf(stderr, "Out of memory -- exit\n");
            exit(1);
        }
        if (*PValue == 0)  {            // Проверка дублирования
            Dups++;
            printf("Duplicate lines %lu:%lu:%s", *PValue, LineNumb, Index);
        } else {
            *PValue = LineNumb;         // Сохранение номера строки
        }
    }
    printf("%lu Duplicates, free JudyHS array of %lu Lines\n",
                    Dups, LineNumb - Dups);
    JHSFA(Bytes, PJArray);              // Освобождение массива JudyHS
    printf("JudyHSFreeArray() free'ed %lu bytes of memory\n", Bytes);
    return (0);
}
```

Функции JudyHS

Функции JudySL обеспечивают библиотеку C для создания и работы с динамическими массивами из массивов байтов размера **Length** с использованием индексов типа word.

Для каждой функции имеется эквивалентный макрос. Поскольку макросы иногда быстрее и проще в обработке ошибок, они являются предпочтительными. Определения функций приведены здесь лишь для полноты.

Одной из сложностей, связанных с вызовами функций JudyHS, является решение вопроса о передаче указателя или его адреса. Поскольку функции, меняющие массив JudyHS, должны менять и указатель на него, нужно передавать

адрес указателя, а не сам указатель. Это ведет к трудно обнаруживаемым ошибкам в программах. На практике макросы позволяют компилятору обнаруживать программные ошибки при передаче указателей вместо адресов.

При вызове функций JudyHS используется дополнительный параметр по сравнению с соответствующими макросами. Этим параметром является указатель на структуру ошибки или **NULL** (если детали ошибок не возвращаются).

Ниже функции описаны в терминах их использования макросами (для случая **#define JUDYERROR_NOTEST 1**). Это является рекомендуемым использованием макросов после завершения отладки программ. Если макрос **JUDYERROR_NOTEST** не задан, объявляется структура ошибки для хранения информации, возвращаемой из функций JudyHS при возникновении ошибок.

Следует обращать внимание на размещение символа **&** в разных функциях.

Синтаксис

```
PPvoid_t JudyHSIns(PPvoid_t PPJHS, void *Index, Word_t Length, PJError_t PJError);
int      JudyHSDel(PPvoid_t PPJHS, void *Index, Word_t Length, PJError_t PJError);
PPvoid_t JudyHSGet(Pcvoid_t  PJHS, void *Index, Word_t Length, PJError_t PJError);
Word_t   JudyHSFreeArray(PPvoid_t PPJHS, PJError_t PJError);
```

Определения

JudyHSIns(&PJHS, Index, Length, &JError)

```
#define JHSI(PValue, PJHS, Index) \
   PValue = JudyLIns(&PJHS, Index, PJE0)
```

JudyHSDel(&PJHS, Index, Length, &JError)

```
#define JHSD(Rc_int, PJHS, Index, Length) \
   Rc_int = JudyHSDel(&PJHS, Index, Length, PJE0)
```

JudyHSGet(PJHS, Index, Length)

```
#define JHSG(PValue, PJHS, Index, Length) \
   PValue = JudyHSIns(PJHS, Index, Length)
```

JudyHSFreeArray(&PJHS, &JError)

```
#define JHSFA(Rc_word, PJHS) \
   Rc_word = JudyHSFreeArray(&PJHS, PJE0)
```

Определения всех функций Judy, типов **Pvoid_t**, **Pcvoid_t**, **PPvoid_t**, **Word_t**, **JError_t** и **PJError_t**, констант **NULL**, **JU_ERRNO_***, **JERR** и **PJE0** приведены в файле **Judy.h** (/usr/include/Judy.h). Следует отметить, что вызывающие должны определять массивы Judy1 типа **Pvoid_t**, который можно передать функциям, принимающим **Pcvoid_t** (constant **Pvoid_t**), а также по адресу в функции, принимающие **PPvoid_t**.

Большинство функций **JudyHS** возвращает **PPvoid_t**, поскольку значения в массиве могут быть ссылками на другие объекты, или **Word_t ***, когда нужен указатель на **Value**, а не указатель на указатель.

Перевод на русский язык

Николай Малых

nmalykh@protocols.ru