A C++ Interface to HBOOK

Benno List

20.11.1998

http://home.cern.ch/~blist/c++/hbookoo.ps

This note describes two interfaces that allow to call HBOOK routines from C++: The first part describes a simple wrapper file that allows to translate FORTRAN programs that use HBOOK basically verbatim to C++. The second part describes a more "object oriented" interface to HBOOK that defines objects which represent histograms, n–tuples, files etc.

Comments, corrections etc. to
Benno.List@cern.ch.

Contents

1 An HBOOK Wrapper File **4**
- 1.1 The C Part of the Wrapper File 4
- 1.2 The C++ Part of the Wrapper File 5
  - 1.2.1 General Policy for the C++ Wrapper Functions 7
  - 1.2.2 Passing of String Arrays and Strings Returned by HBOOK Routines 8
- 1.3 Definition of the `PAWC` Common Block 8
  - 1.3.1 Executive Summary 8
  - 1.3.2 More Detailed Description 9

2 An Object Oriented Interface to HBOOK **12**
- 2.1 Some Internal Details 13
  - 2.1.1 Class `HBEntity` 13
  - 2.1.2 Class `HBHisto` 15
  - 2.1.3 Class `HBHist1` 16
  - 2.1.4 Class `HBHist2` 16
- 2.2 Treating Histograms as Containers: STL Style Functions 16

3 Examples **18**
- 3.1 Examples for the Simple C++ Wrapper 18
  - 3.1.1 Program `HSIMPLE`: Example how to Produce Simple Histograms 18
  - 3.1.2 Program `HTEST`: Advanced Histogram Booking and Filling 18
- 3.2 Examples for the Object Oriented Interface to HBOOK 20
  - 3.2.1 Program `HSIMPLE`: Example how to Produce Simple Histograms 20
  - 3.2.2 Program `HTEST`: Advanced Histogram Booking and Filling 21

4 Reference Manual for the C++ Wrapper File **23**
- 4.1 One and two dimensional histograms - basics 23
  - 4.1.1 Booking 23
  - 4.1.2 Filling 23
  - 4.1.3 Editing 23
  - 4.1.4 Copy, rename, reset and delete 23
- 4.2 Ntuples 24
  - 4.2.1 Row-wise-ntuples (RWN) 24
  - 4.2.2 More general ntuples: column-wise-ntuples (CWN) 24
  - 4.2.3 Making projections of a RWN 27
  - 4.2.4 Get information about an ntuple 27
  - 4.2.5 Ntuple operations 29
- 4.3 Advanced features for booking and editing operations 29
  - 4.3.1 Overview of booking options 29
  - 4.3.2 Filling operations 32
  - 4.3.3 Editing operations 33
- 4.4 Accessing Information 35

4.4.1 Testing if a histogram exists in memory 35
4.4.2 List of histograms 35
4.4.3 Number of entries 35
4.4.4 Histogram attributes contents 35
4.4.5 Contents 36
4.4.6 Errors 36
4.4.7 Associated function 36
4.4.8 Abscissa to channel number 37
4.4.9 Maximum and minimum 37
4.4.10 Rebinning 37
4.4.11 Integrated contents 37
4.4.12 Histogram definition 37
4.4.13 Statistics 38
4.5 Operations on Histograms 38
4.5.1 Arithmetic operations 38
4.5.2 Statistical differences between histograms 38
4.5.3 Bin by bin histogram comparisons 38
4.6 Fitting, parametrization and smoothing 39
4.6.1 Fitting 39
4.6.2 Deprecated fitting routines 40
4.6.3 Parametrization 41
4.6.4 Smoothing 41
4.6.5 Random number generation 42
4.6.6 Fitting with Monte Carlo statistics 42
4.7 Memory management and input/output routines 42
4.7.1 Memory size control 42
4.7.2 Directories 43
4.7.3 Input/output routines 43

References **46**

1 An HBOOK Wrapper File

This wrapper file, which can be found in
`/afs/cern.ch/user/b/blist/public/c++/hbook/hbook.h`,
contains the necessary declarations for all HBOOK routines defined in the HBOOK manual [CER95], except the deprecated routines in section 7.3 "Deprecated fitting routines", and the routine `HNTVDEF` (see section 3.5.2 "Retrieve the contents of a CWN into a common block" of the manual), which is not described in the manual precisely enough to include it into the wrapper file.

The wrapper file is written such that it can be used from ANSI C, in which case it only defines prototypes for the various FORTRAN functions and uses pointers, not references, to pass the arguments of the functions to FORTRAN. Also, only C style comments are used in this section of the wrapper file. All the prototypes have a trailing underscore, as usual when FORTRAN routines are called from C/C++, and the names are in lowercase letters (e.g. `hbook1_`).

If a C++ compiler is used, the compiler or the user has to define the preprocessor macro `__cplusplus`, which is usually done by C++ compilers. If this macro is present, then all the FORTRAN function prototypes are declared `extern"C"`, as necessary in C++. Also, for each HBOOK function on or more C++ wrappers are defined with C++ linkage and a name without trailing underscore (e.g. `hbook1`). These wrapper functions generally pass simple arguments (integers, reals) by value, or, if they are declared as output variables in the HBOOK manual, by reference. For character strings, the length is automatically evaluated, as required by FORTRAN.

The header file contains a guard against multiple inclusion, i.e. is enclosed in:

```
#ifndef H_HBOOK
#define H_HBOOK

/* All the code comes here */

#endif /* ifdef H_HBOOK */
```

1.1 The C Part of the Wrapper File

In the context of C, only the header files `hbook.h` and `FTypes.h` are necessary. The file `FTypes.h` provides various `typedef`s that make the header files more portable. The file `hbook.h` declares function prototypes for (almost) all available HBOOK routines, which generally look like this:

```
#include "FTypes.h"

#ifdef __cplusplus
extern "C" {
#endif /* ifdef __cplusplus */

/* ======================================================= */
/* Chapter 2: One and two dimensional histograms - basics    */
```

```
/* ========================================================== */

/* 2.1 Booking */

/* 2.1.1 One-dimensional case */

  void hbook1_ (FInteger *id,
                const FCharacter *chtitl,
                FInteger *nx,
                FReal *xmi,
                FReal *xma,
                FReal *vmx,
                FStrLength chtitl_length);

/* ...much, much more... */

#ifdef __cplusplus
} // extern "C"
#endif /* ifdef __cplusplus */
```

The function prototypes are ordered according to the sequence in which they are described in the HBOOK manual, and the arguments have exactly the same names, so that the header file itself provides a limited amount of documentation. Note that the length of any character string that is passed to FORTRAN must be passed by value as additional argument. This is done automatically in the C++ wrapper functions.

1.2 The C++ Part of the Wrapper File

The C++ part of the wrapper file defines generally several functions for each FORTRAN HBOOK function, so that it is advisable to check the header file how the functions can be called.

In all cases one declaration is provided that allows a direct, verbatim translation of FORTRAN code to C++, and if strings are passed to the function, it is always possible to pass a C–style string, i. e. a `char *`, or a true C++ `string`:

```
#ifdef __cplusplus
// =============================================================
// Chapter 2: One and two dimensional histograms - basics
// =============================================================

// 2.1 Booking

// 2.1.1 One-dimensional case

inline void hbook1 (FInteger id,
                    const FCharacter *chtitl,
                    FInteger nx,
                    FReal xmi,
                    FReal xma,
```

```
                    FReal vmx = 0) {
  hbook1_ (&id, chtitl, &nx, &xmi, &xma, &vmx, strlen (chtitl));
}
inline void hbook1 (FInteger id,
                    const string& chtitl,
                    FInteger nx,
                    FReal xmi,
                    FReal xma,
                    FReal vmx = 0) {
  hbook1_ (&id, chtitl.c_str(), &nx, &xmi, &xma, &vmx, chtitl.length());
}
#endif /* ifdef __cplusplus */
```

Most of the wrappers do little or no conversions, and are just a trivial syntactic wrapper that need not generate any code at all, e. g.

```
inline void hfill (FInteger id,
                   FReal x,
                   FReal y = 0,
                   FReal weight = 1) {
  hfill_ (&id, &x, &y, &weight);
}
```

Therefore, most of the C++ wrapper functions are directly defined in the header file itself and consequently inlined. Some functions, however, need to do som etransformations of input or output parameters. These are mostly functions which have to do with the passing of string arrays, or where string arguments are returned from FORTRAN, e. g.

```
void hgive  (FInteger id,
             string& chtitl,
             FInteger& nx,
             FReal& xmi,
             FReal& xma,
             FInteger& ny,
             FReal& ymi,
             FReal& yma,
             FInteger& nwt,
             FInteger& loc);
```

These functions are defined in `hbook.cxx`, e. g.

```
void hgive  (FInteger id,
             string& chtitl,
             FInteger& nx,
             FReal& xmi,
             FReal& xma,
             FInteger& ny,
             FReal& ymi,
             FReal& yma,
             FInteger& nwt,
             FInteger& loc) {
```

```
  FCharacter chtitl_ [80];
  hgive_ (&id, chtitl_, &nx, &xmi, &xma, &ny, &ymi, &yma, &nwt, &loc, 80);
  string_from_fortran (80, chtitl_, chtitl);
}
```

Therefor C++ programs that use these wrapper files have to link to `hbook.o`. Also some helper functions are defined in `hbook.cxx`, for instance `string_from_fortran`, which is used in the above example.

1.2.1 General Policy for the C++ Wrapper Functions

Basically the user has to look into the wrapper file in order to find out exactly for each function which ways are available to call it. It was tried to faciliate this by arranging all functions according to the exact order in which they are described in the HBOOk manual, and by inserting the relevant chapter titles as comments in the header file.

However, it was tried to follow a set of rules to give the wrapper file a common "look–and–feel", and to make it easier for the user to guess which wrappers are available, and at the same time make reading of the code produced by the user easier as well.

The following policies were incorporated:

- **Define obvious default values where appropriate.**

  In many cases HBOOK routines require arguments that are needed only in spacial cases. For example, `hfill` expects four arguments:
  `void hfill (FInteger id, FReal x, FReal y, FReal weight);`
  but the weight is in many cases 1 (and consequently often forgotten in FORTRAN), and the value of y is needed only for two–dimensional histograms. Therefore, these parameters have default values:
  `void hfill (FInteger id, FReal x, FReal y = 0, FReal weight = 1);`

  The histogram id is normally a bad candidate for default values, a default value is therefore given only in cases where "0" means "all histograms", as for
  `void hprint (FInteger id = 0).`

  String options, in most cases called `chopt`, often have the empty string as a default value `" "`, e. g. in
  `void hcopy (FInteger id1, FInteger id2, const FCharacter *chtitl=" ");`

- **For each function that takes strings as arguments, provide a wrapper that takes C–style strings, and one that takes C++ strings.**

  This rule was observed for all functions, e. g.:
  `void hcopy (FInteger id1, FInteger id2, const FCharacter *chtitl=" ");`
  `void hcopy (FInteger id1, FInteger id2, const string& chtitl);`

  Note that the function that takes a `string` as argument does not provide a default argument for `chtitl`, because that would make the call

```
hcopy (10, 20);
```
ambiguous.

- **Functions that have exactly one output variable return the value of this variable as function result, and provide a function form which omits the passing of the output variable in the parameter list.**

  For example, for `hix` the following wrappers are provided:

```
inline FReal hix   (FInteger id, FInteger i, FReal& x) {
  hix_ (&id, &i, &x);
  return x;
}
inline FReal hix   (FInteger id, FInteger i) {
  FReal x;
  hix_ (&id, &i, &x);
  return x;
}
```

  Therefore, `hix` can be used in the following ways:

```
void f (int id, int i) {
  FReal x1;
  hix (id, i, x1);  // x1 passed by reference, return value of hix
                    // is discarded;x1 must be of type FReal, because
                    // it is passed  by reference.
                    // id and i are input values and passed by value,
                    // therefore they need not be of type FInteger.
  float x2;
  x2 = hix (id, i); // if FReal is not equivalent to type float,
                    // an automatic conversion will be done.
}
```

  The first use of `hix` is the verbatim translation of the FORTRAN way to call it, but the meaning of the second call should be immediately clear to everyone, probably it is even more transparent, because it can be immediately seem which one is the output argument.

1.2.2 Passing of String Arrays and Strings Returned by HBOOK Routines

1.3 Definition of the `PAWC` Common Block

1.3.1 Executive Summary

You define the size of the `PAWC` common in the include file `nhmem.h`:

```
// file nhmem.h

#ifndef H_NHMEM
```

```
#define H_NHMEM
// edit this line to set size of HBOOK store
enum {NHMEM =  20000};      // Size of HBOOK store
#endif
```

If you have changed `nhmem.h`, you have to recompile `hbook.cxx` and relink your program.

You do not have to declare or define the `PAWC` common block anywhere, neither do you have to call `hlimit`, all this is done automatically when you include `nhmem.h` into your source files and link with `hbook.o`.

1.3.2 More Detailed Description

The `PAWC` common block is basically only a big chunk of memory that can be used by HBOOK. The size of this common is communicated to HBOOK by a call to `hblimit`, which must be done before the first use of any HBOOK routine.

The size of the `PAWC` common is defined by the constant , which is defined in an extra header file `nhmem.h`, to make it easier for the user to change the common size:

```
// file nhmem.h

#ifndef H_NHMEM
#define H_NHMEM
// edit this line to set size of HBOOK store
enum {NHMEM =  20000};      // Size of HBOOK store
#endif
```

This header is included in `hbook.h`:

```
// file hbook.h

#ifndef H_HBOOK
#define H_HBOOK

#include "FTypes.h"
#ifdef __cplusplus
  #include "nhmem.h"


//======================================================================
// The PAWC common

// For a discussion of the initialization technique used here, see
// B. Stroustrup, The C++ programming language, 3rd ed., p. 639.

struct PAWC_Type {
  FInteger memor[NHMEM];

  struct Init {
    Init() {
      if (count++ == 0) {
```

```
      hlimit (NHMEM);
    }
  }

  ~Init() {
    --count;
  }

  static int count;
 };
};

static PAWC_Type::Init pawc_init_;

#endif /* ifdef __cplusplus */
```

Finally, the static variable `PAWC_Type::Init::count` has to be defined (as opposed to merely declared) in some compilation unit, as well as the PAW common itself, which is done in `hbook.cxx`:

```
// file hbook.cxx

#include "hbook.h"

//====================================================================
// The PAWC common

// Definition of the PAWC common

PAWC_Type pawc_;

int PAWC_Type::Init::count = 0;
```

What does all this do?

Obvious enough, the line
`PAWC_Type pawc_;`
in `hbook.cxx` creates the PAWC common, which is basically defined as one big array of size `NHMEM`:

```
struct PAWC_Type {
  FInteger memor[NHMEM];
};
```

However, it would be nice to call `hlimit` at the time the common is created. One obvious way to achieve this would be:

```
struct PAWC_Type {
  FInteger memor[NHMEM];

  PAWC_Type() {
    hlimit (NHMEM);      // hlimit called in constructor
```

```
  }
};
```

This would work, but not if someone wrote code like this:

```
// in file utility.h:
#include "hbook.h"

class Monitor {
public:
  Monitor (int theId): id (theId) {
    hbook (id, "Monitor histogram", 100, 0, 100);
  }
  ~Monitor() {
    hprint (id);
  }
private:
  int id;
};

class SomeUtilityClass {
public:
  SomeUtilityClass (int i) {
    hfill (1, i);   // do some monitoring of produced objects
  }
// other, more useful member functions

private:
  static Monitor m;
};

// in file utility.cxx:
#include "utility.h"              /* hbook.h included via utility.h */

Monitor SomeUtilityClass::m (1);  // creates an Monitor object,
                                  // which in turn creates a
                                  // HBOOK histogram
```

Now, when `utility.o` is linked into some program, before the start of the main program the `Monitor` object `m` is created, which books some histogram. This histogram is used to monitor the use of the class `SomeUtilityClass`, and printed out at the end of the program, when `m`'s destructor is called (try it!).

To achieve this, `Monitor`'s constructor calls `hfill`, which is only possible if `hlimit` has already be called.

However, `hlimit` is called in the present scheme at some arbitrary time, the only guarantee being that it is called before `main` starts. This is not enough.

The way it is really done in `hbook.h` solves the problem: through `utility.h`, `hbook.h` is included in `utility.cxx`. Now `utility.h` contains a statement
`static PAWC_Type::Init pawc_init_;`
which creates a variable of type `PAWC_Type::Init` with internal linkage, i.e. it is accessible only from the code within the file `utility.cxx`, as indicated by the keyword `static`.

The creation of `pawc_init_`, through its constructor, now triggers the call to `hlimit`, if the `counter` is at 0, i. e. if no other instance of type `PAWC_Type::Init` has yet been created. Therefore, `hlimit` is guarantedd to have been properly called before the call to `hfill`.

This technique is commonly used in the initialization of libraries and discussed in [Str97, p. 639] and [Sch96].

2 An Object Oriented Interface to HBOOK

In order to allow an object oriented programming style, classes are needed that correspond to histograms, n-tuples etc. in HBOOK.

An effort was started to define such classes. The relevant files are located in `/afs/cern.ch/user/b/blist/public/c++/hbook/`,
and are named
`hbookoo.h`,
`hbookoo.cxx`
and
`fortclose.h`,
`fortclose.f`,
where `fortclose.f` just contains a simple FORTRAN routine which issues a FORTRAN `CLOSE` statement for a given FORTRAN unit.

The following classes are available at the moment:

- `class HBHist1` for one–dimensional histograms.
- `class HBHist2` for two–dimensional histograms.
- `class HBHisto`, an abstract base class of classes `class HBHist1` and `class HBHist2` that captures the common features of one– and two–dimensional histograms.
- `class HBEntity`, an abstract base class of class `class HBHisto` (in future, the n–tupel classes will be derived from this class as well) that captures the common behaviour of all HBOOK objects, i. e. the fact that they are characterized by an ID number.

A basic design consideration for these classes was that they should not "mirror" any information that is stored with the HBOOK histograms and can be found out by calling HBOOK functions, like bin contents, titles etc. This is to avoid problems that arise from inconsistencies between what is really stored in the PAWC common and what is stored with the HBOOK objects. Therefore, basically only the ID is stored with the objects[1].

From the abovementioned classes, two kinds of objects can be instantiated: One- and two-dimensional histograms. After an object has been created, the histogram ID is not needed anymore, because the histogram objects represents

[1] Subdirectories are not yet supported in this framework. To support them would probably make it necessary also to store the directory name with an histogram object.

the histogram in memory. It is important that the histogram object "knows" whether the object with a certain ID is a one– or two-dimensional histogram, an n–tuple, or does not exist at all.

In order to avoid that two different objects are created that correspond to the same ID, the constructors of `class HBHist1` and `class HBHist2` are private, and histograms can only be accessed through pointers (the recommended way) or references to histogram objects:

```
#include "hbookoo.h"
void f() {
  HBHist1 *hist1 = HBHist1::hbook1 (10, "1-DIM HISTOGRAM", 100, 1, 101);
  HBHist2 *hist2 = HBHist2::hbook2 (20, "SCATTER-PLOT", 100, 0, 1, 40, 1, 41, 30);

  if (hist1 != 0) {                 // check whether histogram was created
    hist1->hfill (1);               // x = 1, weight = 1
    hist1->hfill (2.7, 3.5);        // x = 2.7, weight = 3.5
  }
  if (hist2 != 0) {                 // check whether histogram was created
    hist2->hfill (0.7, 3.5);        // x = 0.7, y = 3.5, weight = 1
    hist2->hfill (1, 2, 3.14);      // x = 1, y = 2, weight = 3.14
  }
}
```

If a histogram could not be created, the functions `HBHist1::hbook1` and `HBHist2::hbook2` return null pointers, which is why the use of references is discouraged.

2.1 Some Internal Details

To prevent the user from creating two different histogram objects with the same ID, a list is kept in class `HBEntity` of all such objects. Actually, it is a `map`, which maps the ID number to a pointer to the corresponding object.

If no object exists to a histogram with a given ID, even if the histogram exists in the PAWC memory, it is allowed to create a new histogram object with that ID, thereby overwriting the old histogram. This is consistent with the way this is handeled in FORTRAN. Only the coexistence of two objects referring to the same ID must be prevented, because this clearly corresponds to a programming error and will result in unwanted behaviour of the program. This is the reason why `HBEntity` keeps a list of existing histogram objects, rather than to check with an `hexist(id)` call if the histogram ID has already been used.

2.1.1 Class `HBEntity`

This is the core of class `HBEntity`:

```
// file hbookoo.h

#ifndef H_HBOOKOO
#define H_HBOOKOO
```

```
#include <map>
#include "hbook.h"

class HBEntity {
public:
  int getID () const;
  static HBEntity *getHBEntity (int theID);

  typedef map<int, HBEntity *> MapType;
  typedef MapType::iterator MapIterator;

  virtual ~HBEntity();

protected:
  explicit HBEntity (int newID);

private:
  HBEntity (const HBEntity&);                    // no copy constructor
  const HBEntity& operator= (const HBEntity&);   // no assignment

  static MapType entityList;

// Data members:
  int id;
};
#endif

// file hbookoo.cxx:

#include "hbook.h"
#include "hbookoo.h"
#include<map>

HBEntity::MapType HBEntity::entityList; // definition of entityLists

HBEntity *HBEntity::getHBEntity (int theID) {
  return entityList[theID];
}

HBEntity::HBEntity (int newID) {
  id = newID;
  entityList [id] = this;
}

HBEntity::~HBEntity() {
  entityList.erase (id);
}
```

The construtor of `HBEntity` is protected, so no instances of this class can be created by the user. This class just stores the ID number of any HBOOK object, and administrated the list of all such objects in use.

It is also possible to get a pointer to an object, which already exists, by means of its ID, through the function `getHBEntity (int theID)`.

Additionally, `HBEntity` can define member functions that can be applied to all HBOOK objects, whether they are histograms or n–tuples. At the moment, this is only `htitle`.

2.1.2 Class `HBHisto`

Class `HBHisto` is an abstract class that defines some functions which can be applied to all histograms, like `hprint()`, and some purely virtual functions which exist for both, one– and two–dimensional histograms, but work differently on them. An example for the latter type of functions is `hbar()`, which means "store all errors for this histogram", and translates to `hbarx()` for one–dimensional histograms, and `hbar2()` for two–dimensional histograms.

Some code for this class is shown below:

```
// file hbookoo.h

#include "hbook.h"

class HBHisto: public HBEntity {
public:
  virtual HBHisto *hcopy (int newID, const FCharacter *chtitle) = 0;

  inline void hprint() const;

  virtual void hbar() = 0;
  virtual void hbarx();

  static inline void histdo();

protected:
  explicit HBHisto (int newID);

private:
  const HBHisto& operator= (const HBHisto&);     // no assignment
};

// file hbookoo.cxx
#include "hbook.h"
#include "hbookoo.h"
#include <iostream.h>

HBHisto::HBHisto (int theID)
  : HBEntity (theID)
{}

void HBHisto::hbarx()
{
  ::hbarx (getID());
}
```

2.1.3 Class HBHist1

2.1.4 Class HBHist2

2.2 Treating Histograms as Containers: STL Style Functions

In a certain sense, a histogram can be viewed as an ordered collection of bins, where each bin contains sone information, especially the bin content, the error, and the left and right bin boundaries. In many cases it is necessary to do something to all bins, for instance to normalize the bin contents, to sum them up etc. Therefore it may be useful to treat the histogram as container (like a STL `vector`) of bin, which provides iterators and thereby makes it possible to apply STL algorithms.

Here is an example:

```
#include "hbookoo.h"
#include <numeric>     /* defines accumulate */
#include <algorithm>   /* defines transform, find_if */
#include <vector>      /* defines vector */
#include <functional>  /* defines bind2nd, greater<T> */

class IntegrateNormalize {
public:
  double operator() (double x) {
    return (runningsum += x*normalization);
  }

  explicit IntegrateNormalize (double sum = 1, double norm = 1, double start = 0)
  : normalization (norm/sum), runningsum (start)
  {}

  double normalization;
  double runningsum;
};

void f(HBHist1 *hist1, HBHist1 *hist2) {
  if (hist != 0) {
    vector<float> contents (hist1->contents().begin(), hist1->contents().end());

    float sum = accumulate (hist1->contents().begin(), hist1->contents().end(), 0.);

    vector<float> summedcontents (hist1->contents().size());
    transform (hist1->contents().begin(), hist1->contents().end(),
           summedcontents.begin(),
           IntegrateNormalize (sum));

    hist2->contents() = summedcontents;
```

```
    HBHist1ContentsIterator bin = find_if (hist2->contents.begin(),
                                           hist2->contents.end(),
                                           bind2nd (greater<float>(), 0.2));
  }
}
```

Obviously, `hist1->contents()` can be used like any STL container, which contains the bin contents of the histogram. It provides iterators, especially the iterators `hist1->contents().begin()` and `hist1->contents().end()`, which point to the (contents of the) first bin and the bin after the last bin, respectively.

These iterators can be used to fill a vector, or to run STL algorithms, like `accumulate`, `transform` or `find_if`.

Of course, the above example looks strange at first sight. However, for someone who is used a bit to the STL, it should be immediately apparent what the various statements do:

Function `f` is passed two pointers to one–dimensional histograms. If this pointer is non–zero, a vector `contents` is filled with the bin contents of the histogram. Then all bins are summed up with `accumulate`[2].

After that, a function object `IntegrateNormalize` is run over the bins, whoose output is stored into the newly created vector `summedcontents`. The function object successively sums up all channel contents, and returns the running sum, normalized to some number it is given at the beginning, in this case the summed contents of the histogram. Observe that `IntegrateNormalize` works with doubles, although the bin contents are of type `FReal`, which on most machines will be `float`. The automatic type–conversion of C++ takes care of that. This would not be possible if one would use a function, rather than a function object, for this kind of task.

These integrated and normalized contents are then put into another histogram.

Finally, the first bin in this histogram is searched that has a content greater than 0.2.

To do all this, a number of classes are necessary. At the moment, the following classes exist:

- `class HBHist1Contents` provides objets that look like a container with the bin contents of a histogram.
- `class HBHist1ContentsIterator` provides random–access iterators for the bin contents.
- `class HBHist1Errors` provides objets that look like a container with the bin errors of a histogram.
- `class HBHist1ErrorsIterator` provides random–access iterators for the bin errors.

[2]Yes, I know that this could be easily done with `hsum()`.

3 Examples

3.1 Examples for the Simple C++ Wrapper

3.1.1 Program HSIMPLE: Example how to Produce Simple Histograms

This example is a verbatim translation of the example program given on page 5 of the HBOOK manual [CER95]:

```
#include <cstring>
#include <cmath>
#include "hbook.h"

// Example program from page 5 of the HBOOK manual

// Benno List, Benno.List@cern.ch
// 17.11.98

int main () {
     htitle ("EXAMPLE NO = 1");

//   Book 1-dim histogram and scatter-plot
     hbook1 (10, "EXAMPLE OF 1-DIM HISTOGRAM", 100, 1, 101);
     hbook2 (20, "EXAMPLE OF SCATTER-PLOT", 100, 0, 1, 40, 1, 41, 30);

//   Fill 1-dim histogram
     for (int i = 1; i <= 100; i++) {
       float w = 10.*(i % 25);
       hfill (10, i+0.5, 0, w);
     }

//   Fill scatter-plot
     float x = -0.005;
     for (int i = 1; i <= 100; i++) {
       x += 0.01;
       for (int j = 1; j <= 40; j++) {
         int iw = (i % 25)*(j % 10);
         int iwmax = j - (i % 25) + 10;
         if (iw > iwmax) iw = 0;
         hfill (20, x, j, iw);
       }
     }

     histdo();

     return 0;
}
```

3.1.2 Program HTEST: Advanced Histogram Booking and Filling

This example is a verbatim translation of the example program given on page 9 of the HBOOK manual [CER95]:

```
#include <cmath>
#include "hbook.h"

// Example program from page 9 of the HBOOK manual

// Benno List, Benno.List@cern.ch
// 17.11.98

float htfun1 (const float& x) {

     const float c1 = 1;
     const float c2 = 0.5;
     const float xm1 = 0.3;
     const float xm2 = 0.7;
     const float xs1 = 0.07;
     const float xs2 = 0.12;

     float a1 = -0.5*pow (double((x - xm1)/xs1), 2);
     float a2 = -0.5*pow (double((x - xm2)/xs2), 2);
     float x1 = (fabs (a1) > 0.0001) ? (c1*exp (a1)) : c1 ;
     float x2 = (fabs (a2) > 0.0001) ? (c2*exp (a2)) : c2 ;

     return x1 + x2;
}

float htfun2 (const float& x, const float& y) {
     return htfun1 (x)*htfun1 (y);
}

int main () {
     hbfun1 (100, "Test of HRDM1", 100, 0, 1, &htfun1);
     hbook1 (110, "Filled according to HTFUN1", 100, 0, 1, 1000);

     hbfun2 (200, "Test of HRDM2", 100, 0, 1, 40, 0, 1, &htfun2);
     hscale (200, 0);
     hbook2 (210, "Filled according to HTFUN2", 100, 0, 1, 40, 0, 1, 30);

     for (int i = 0; i < 1000; i++) {
       float x = hrndm1 (100);
       hfill (110, x, 0, 1);
       float y;
       hrndm2 (200, x, y);
       hfill (210, x, y, 1);
     }

     hrput (0, "htest.dat", "N");
     hdelet (100);
     hdelet (200);
     hprint (0);

     return 0;
}
```

3.2 Examples for the Object Oriented Interface to HBOOK

3.2.1 Program `HSIMPLE`: Example how to Produce Simple Histograms

This example is a object oriented style translation of the example program given on page 5 of the HBOOK manual [CER95]:

```
#include <cstring>
#include <cmath>
#include "hbookoo.h"
#include "hbook.h"
#include<iostream.h>

// Example program from page 5 of the HBOOK manual

// Benno List, Benno.List@cern.ch
// 17.11.98

// The hbook store is declared in hbook.h and defined in hbook.cxx
// its size is defined in nhmem.h

// You have to link with hbook.o, hbookoo.o and fortclose.o
// to make this program work


int main () {
     HBEntity::htitle ("EXAMPLE NO = 1");

//   Book 1-dim histogram and scatter-plot
     HBHist1 *hist1 = HBHist1::hbook1 (10, "EXAMPLE OF 1-DIM HISTOGRAM",
                                       100, 1, 101);
     HBHist2 *hist2 = HBHist2::hbook2 (20, "EXAMPLE OF SCATTER-PLOT",
                                       100, 0, 1, 40, 1, 41, 30);

//   Fill 1-dim histogram

     for (int i = 1; i <= 100; i++) {
       float w = 10.*(i % 25);
       hist1->hfill (i+0.5, w);
     }

//   Fill scatter-plot
     float x = -0.005;
     for (int i = 1; i <= 100; i++) {
       x += 0.01;
       for (int j = 1; j <= 40; j++) {
       int iw = (i % 25)*(j % 10);
       int iwmax = j - (i % 25) + 10;
       if (iw > iwmax) iw = 0;
       hist2->hfill (x, j, iw);
       }
     }
```

```
    HBHisto::histdo();
}
```

3.2.2 Program HTEST: Advanced Histogram Booking and Filling

This example is a object oriented translation of the example program given on page 9 of the HBOOK manual [CER95]. One difference compaired to the non–object oriented example (see section 3.1.2) is the use of function objects rather than function pointers in conjunction with `hrndm1`. Function pointers could be used instead, but function objects are more versatile, because it is easier to "configure" them (in this example, the parameters of the user function can be set during the creation of the function object), and because the function object can have types different from `FReal` as arguments and return type.

```
#include <cmath>
#include "FTypes.h"
#include "hbookoo.h"
#include <vector>

// Example program from page 9 of the HBOOK manual

// Benno List, Benno.List@cern.ch
// 17.11.98

class Htfun1 {
public:
  double operator() (double x) {     // function object may take and return doubles
                                     // transformation is done by wrapper function!
     float a1 = -0.5*pow (double((x - xm1)/xs1), 2);
     float a2 = -0.5*pow (double((x - xm2)/xs2), 2);
     float x1 = (fabs (a1) > 0.0001) ? (c1*exp (a1)) : c1 ;
     float x2 = (fabs (a2) > 0.0001) ? (c2*exp (a2)) : c2 ;

     return x1 + x2;
  }

  Htfun1 (float c1_ = 1,
          float c2_ = 0.5,
          float xm1_ = 0.3,
          float xm2_ = 0.7,
          float xs1_ = 0.07,
          float xs2_ = 0.12)
  : c1 (c1_), c2 (c2_), xm1 (xm1_), xm2 (xm2_), xs1 (xs1_), xs2 (xs2_)
  {}

  float c1;
  float c2;
  float xm1;
  float xm2;
  float xs1;
  float xs2;
};
```

```
class Htfun2 {
public:
  double operator() (double x, double y) {
    return htfun1a (x)*htfun1b (y);
  }
  Htfun2 (Htfun1 a = Htfun1(), Htfun1 b = Htfun1())
  : htfun1a (a), htfun1b (b)
  {}

  Htfun1 htfun1a;
  Htfun1 htfun1b;
};

int main () {
     HBHist1 *hist1  = HBHist1::hbfun1 (100, "Test of HRDM1", 100, 0, 1, Htfun1());
     HBHist1 *hist1a = HBHist1::hbook1 (110, "Filled according to HTFUN1",
                                         100, 0, 1, 1000);

     HBHist2 *hist2  = HBHist2::hbfun2 (200, "Test of HRDM2",
                                         100, 0, 1, 40, 0, 1, Htfun2());
     hist2->hscale ();
     HBHist2 *hist2a = HBHist2::hbook2 (210, "Filled according to HTFUN2",
                                         100, 0, 1, 40, 0, 1, 30);

     for (int i = 0; i < 1000; i++) {
       FReal x = hist1.hrndm1();
       hist1a->hfill (x);
       FReal y;
       hist2->hrndm2 (x, y);
       hist2a->hfill (x, y);
     }

     hrput (0, "htest.dat", "N");
     hdelet (100);
     hdelet (200);
     hprint (0);
}
```

4 Reference Manual for the C++ Wrapper File

4.1 One and two dimensional histograms - basics

See HBOOK manual chapter 2, page 14.

4.1.1 Booking

See HBOOK manual, section 2.1.

One-dimensional case

See HBOOK manual, section 2.1.1.

```
inline void hbook1 (FInteger id, const FCharacter *chtitl, FInteger nx,
    FReal xmi, FReal xma, FReal vmx = 0);
```

```
inline void hbook1 (FInteger id, const string& chtitl, FInteger nx, FReal
    xmi, FReal xma, FReal vmx = 0);
```

Two-dimensional case

See HBOOK manual, section 2.1.2.

```
inline void hbook2 (FInteger id, const FCharacter *chtitl, FInteger nx,
    FReal xmi, FReal xma, FInteger ny, FReal ymi, FReal yma, FReal vmx = 0);
```

```
inline void hbook2 (FInteger id, const string& chtitl, FInteger nx, FReal
    xmi, FReal xma, FInteger ny, FReal ymi, FReal yma, FReal vmx = 0);
```

4.1.2 Filling

See HBOOK manual, section 2.2.

```
inline void hfill (FInteger id, FReal x, FReal y = 0, FReal weight = 1);
```

4.1.3 Editing

See HBOOK manual, section 2.3.

```
inline void histdo ();
```

```
inline void hprint (FInteger id = 0);
```

4.1.4 Copy, rename, reset and delete

See HBOOK manual, section 2.4.

```
inline void hcopy (FInteger id1, FInteger id2, const FCharacter *chtitl = " ");
```

```
inline void hcopy (FInteger id1, FInteger id2, const string& chtitl);
```

```
inline void hcopypr (FInteger id1, FInteger id2, const FCharacter *chtitl,
    FInteger ibinx1, FInteger ibinx2, FInteger ibiny1 = 0, FInteger ibiny2 = 0);
```

```
inline void hcopypr (FInteger id1, FInteger id2, const string& chtitl, FInteger
    ibinx1, FInteger ibinx2, FInteger ibiny1 = 0, FInteger ibiny2 = 0);
```

```
inline void hcopypr (FInteger id1, FInteger id2, FInteger ibinx1, FInteger
    ibinx2, FInteger ibiny1 = 0, FInteger ibiny2 = 0);
```

This version of `hcopypr` sets `chtitl` to the empty string (i. e. `" "`).

```
inline void hrenid (FInteger idold, FInteger idnew);
```

```
inline void hreset (FInteger id, const FCharacter *chtitl = " ");
```

```
inline void hreset (FInteger id, const string& chtitl);
```

```
inline void hdelet (FInteger id);
```

4.2 Ntuples

See HBOOK manual chapter 3, page 18.

4.2.1 Row-wise-ntuples (RWN)

See HBOOK manual, section 3.2.

Booking a RWN

See HBOOK manual, section 3.2.1.

```
template<int chtagslen>
void hbookn (FInteger id, const FCharacter *chtitl, FInteger nvar, const
    FCharacter *chrzpa, FInteger nwbuff, const FCharacter chtags [][chtagslen]);
```

```
void hbookn (FInteger id, const string& chtitl, FInteger nvar, const string&
    chrzpa, FInteger nwbuff, const string chtags[]);
```

Filling a RWN

See HBOOK manual, section 3.2.2.

```
inline void hfn (FInteger id, const FReal x[]);
```

```
inline void hfnov (FInteger id, const FReal x[]);
```

4.2.2 More general ntuples: column-wise-ntuples (CWN)

See HBOOK manual, section 3.3.

Booking a CWN

See HBOOK manual, section 3.3.1.

```
inline void hbnt (FInteger id, const FCharacter *chfile = " ", const FCharacter
    *chopt = " ");
```

```
inline void hbnt (FInteger id, const string& chfile, const string& chopt = " ");
```

```
inline FInteger hbset (const FCharacter *chopt, FInteger ival, FInteger&
    ierr);
```

All versions of `hbset` return the value of the errorflag `ierr`.

```
inline FInteger hbset (FInteger ival, FInteger& ierr);
```

This version of `hbset` sets `chopt` to the empty string (i. e. `" "`).

```
inline FInteger hbset (const FCharacter *chopt, FInteger ival);
```

```
inline FInteger hbset (FInteger ival = 1024);
```

```
inline FInteger hbset (const string& chopt, FInteger ival, FInteger& ierr);
```

```
inline FInteger hbset (const string& chopt, FInteger ival);
```

Describing the columns of a CWN

See HBOOK manual, section 3.3.2.

```
template <class C>
inline void hbname (FInteger id, const FCharacter *chblok, C& variable,
    const FCharacter *chform);
```

This version of `hbname` can be called with a reference to any kind of variable. It is most commonly used with single variables.

```
template <class C>
inline void hbname (FInteger id, const FCharacter *chblok, C *variable,
    const FCharacter *chform);
```

This version of `hbname` can be called with a pointer to any kind of variable. It is most commonly used with arrays.

```
template <class C>
inline void hbname (FInteger id, const string& chblok, C& variable, const
    string& chform);
```

```
template <class C>
inline void hbname (FInteger id, const string& chblok, C *variable, const
    string& chform);
```

```
template <class C>
inline void hbnamc (FInteger id, const FCharacter *chblok, C& variable,
    const FCharacter *chform);
```

```
template <class C>
inline void hbnamc (FInteger id, const FCharacter *chblok, C *variable,
    const FCharacter *chform);
```

```
template <class C>
inline void hbnamc (FInteger id, const string& chblok, C& variable, const
    string& chform);
```

```
template <class C>
inline void hbnamc (FInteger id, const string& chblok, C *variable, const
    string& chform);
```

Creating CHFORM dynamically

See HBOOK manual, section 3.3.3.

```
inline void hnform (const FCharacter *chform, const FCharacter *chname,
    FInteger ldim, const FCharacter *chtype, FReal xlow = 0, FReal xhigh = 0);
```

```
inline void hnform (const FCharacter *chform, const FCharacter *chname,
    FInteger ldim[], const FCharacter *chtype, FReal xlow = 0, FReal xhigh = 0);
```

```
inline void hnform (const FCharacter *chform, const FCharacter *chname,
    const FCharacter *chtype, FReal xlow = 0, FReal xhigh = 0);
```

This version of `hnform` sets `ldim` to zero.

```
inline void hnform (const string& chform, const string& chname, FInteger
    ldim, const string& chtype, FReal xlow = 0, FReal xhigh = 0);
```

```
inline void hnform (const string& chform, const string& chname, FInteger
    ldim[], const string& chtype, FReal xlow = 0, FReal xhigh = 0);
```

```
inline void hnform (const string& chform, const string& chname, const string&
    chtype, FReal xlow = 0, FReal xhigh = 0);
```

This version of `hnform` sets `ldim` to zero.

```
template <class C, int chtagslen>
void hbooknc (FInteger id, const FCharacter *chtitl, FInteger nvar, const
    FCharacter *block, C& tuple, const FCharacter chtags [][chtagslen]);
```

```
template <class C, int chtagslen>
void hbooknc (FInteger id, const FCharacter *chtitl, FInteger nvar, const
    FCharacter *block, C *tuple, const FCharacter chtags [][chtagslen]);
```

```
template <class C>
void hbooknc (FInteger id, const string& chtitl, FInteger nvar, const string&
    block, C& tuple, const string chtags[]);
```

```
template <class C>
inline void hbooknc (FInteger id, const string& chtitl, FInteger nvar, const
    string& block, C *tuple, const string chtags[]);
```

Filling a CWN

See HBOOK manual, section 3.3.4.

```
inline void hfnt (FInteger id);
```

```
inline void hfntb (FInteger id, const FCharacter *chblok);
```

```
inline void hfntb (FInteger id, const string& chblok);
```

```
inline void hprnt (FInteger id);
```

```
inline void hrecov (FInteger id, const FCharacter *chopt = " ");
```

```
inline void hrecov (FInteger id, const string& chopt);
```

4.2.3 Making projections of a RWN

See HBOOK manual, section 3.4.

```
inline void hproj1 (FInteger id, FInteger idn, FInteger isel, FReal fun
    (const FReal[], const FInteger *), FInteger ifrom, FInteger ito, FInteger
    ivarx);
```

```
inline void hproj1 (FInteger id, FInteger idn, FInteger isel, FReal fun
    (const FReal[], const FInteger&), FInteger ifrom, FInteger ito, FInteger
    ivarx);
```

```
inline void hproj2 (FInteger id, FInteger idn, FInteger isel, FReal fun
    (const FReal[], const FInteger *), FInteger ifrom, FInteger ito, FInteger
    ivarx, FInteger ivary);
```

```
inline void hproj2 (FInteger id, FInteger idn, FInteger isel, FReal fun
    (const FReal[], const FInteger&), FInteger ifrom, FInteger ito, FInteger
    ivarx, FInteger ivary);
```

4.2.4 Get information about an ntuple

See HBOOK manual, section 3.5.

```
template<int chtitllen, int chtaglen>
void hgiven (FInteger id, FCharacter chtitl [chtitllen], FInteger& nvar,
    FCharacter chtag[][chtaglen], FReal rlow[], FReal rhigh[]);
```

This version of `hgiven` gives the histogram title back into a character array declared in the calling function as
`FCharacter chtitl [CHTITLLEN];`
or similar, and the tag names into an array declared in the calling function as
`FCharacter chtag [NVAR][CHTAGLEN];`
or similar. If the resulting strings do not fit into the character arrays, they are chopped off. The resulting string is zero–terminated and trailing spaces are removed.

```
void hgiven (FInteger id, string& chtitl, FInteger& nvar, string chtag[],
    FReal rlow[], FReal rhigh[]);
```

This version of `hgiven` gives the histogram title back into a STL string declared in the calling function as
`string chtitl;`
or similar, and the tag names into a string array declared in the calling function as
`string chtag [NVAR];`
or similar. Trailing spaces in the resulting strings are removed.

Retrieve the contents of a RWN into an array

See HBOOK manual, section 3.5.1.

```
inline FInteger hgn (FInteger id, FInteger& idn, FInteger idnevt, FReal
    x[], FInteger& ierror);
```

All versions of `hgn` return the value of the errorflag `ierror`.

```
inline FInteger hgn (FInteger id, FInteger& idn, FInteger idnevt, FReal
    x[]);
```

```
inline void hgnpar (FInteger id, const FCharacter *chrout = " ");
```

```
inline void hgnpar (FInteger id, const string& chrout);
```

```
inline FInteger hgnf (FInteger id, FInteger idnevt, FReal x[], FInteger&
    ierror);
```

All versions of `hgnf` return the value of the errorflag `ierror`.

```
inline FInteger hgnf (FInteger id, FInteger idnevt, FReal x[]);
```

Retrieve the contents of a CWN into a common block

See HBOOK manual, section 3.5.2.

```
inline FInteger hgnt (FInteger id, FInteger idnevt, FInteger& ierr);
```

All versions of `hgnt` return the value of the errorflag `ierror`.

```
inline FInteger hgnt (FInteger id, FInteger idnevt);
```

```
inline FInteger hgntb (FInteger id, const FCharacter *chblok, FInteger irow,
    FInteger& ierr);
```

All versions of `hgntb` return the value of the errorflag `ierror`.

```
inline FInteger hgntb (FInteger id, const FCharacter *chblok, FInteger irow);
```

```
inline FInteger hgntb (FInteger id, const string& chblok, FInteger irow,
    FInteger& ierr);
```

```
inline FInteger hgntb (FInteger id, const string& chblok, FInteger irow);
```

```
template<int chvarlen>
FInteger hgntv (FInteger id, const FCharacter chvar[][chvarlen], FInteger
    nvar, FInteger irow, FInteger& ierr);
```

All versions of `hgntv` return the value of the errorflag `ierror`.

```
template<int chvarlen>
FInteger hgntv (FInteger id, const FCharacter chvar[][chvarlen], FInteger
    nvar, FInteger irow);
```

```
FInteger hgntv (FInteger id, const string chvar[], FInteger nvar, FInteger
    irow, FInteger& ierr);
```

```
inline FInteger hgntf (FInteger id, FInteger irow, FInteger& ierr);
```

All versions of `hgntf` return the value of the errorflag `ierror`.

```
inline FInteger hgntf (FInteger id, FInteger irow);
```

`hntvdef`

This routine is not sufficiently described in the HBOOK manual to be included in the wrapper file.

Generate a user function

See HBOOK manual, section 3.5.3.

```
inline void huwfun (FInteger lun, FInteger id, const FCharacter *chfun,
    FInteger itrunc = 0, const FCharacter *chopt = "B");
```

```
inline void huwfun (FInteger lun, FInteger id, const string& chfun, FInteger
    itrunc = 0, const string& chopt = "B");
```

4.2.5 Ntuple operations

See HBOOK manual, section 3.6.

```
inline void hntdup (FInteger id1, FInteger id2, FInteger newbuf = 0, const
    FCharacter *chtitl = " ", const FCharacter *chopt = " ");
```

```
inline void hntdup (FInteger id1, FInteger id2, FInteger newbuf, const string&
    chtitl, const string& chopt = " ");
```

```
inline void hrename (FInteger id, const FCharacter *chold, const FCharacter
    *chnew);
```

```
inline void hrename (FInteger id, const string& chold, const string& chnew);
```

```
inline FInteger hconvol (FInteger id1, FInteger id2, FInteger id3, FInteger&
    ierror);
```

All versions of `hconvol` return the value of the errorflag `ierror`.

```
inline FInteger hconvol(FInteger id1, FInteger id2, FInteger id3);
```

4.3 Advanced features for booking and editing operations

See HBOOK manual chapter 4, page 53.

4.3.1 Overview of booking options

See HBOOK manual, section 4.1.

Histograms with non-equidistant bins

See HBOOK manual, section 4.1.1.

```
inline void hbookb (FInteger id, const FCharacter *chtitl, FInteger ncx,
    const FReal xbins[], FReal vmx = 0);
```

```
inline void hbookb (FInteger id, const string& chtitl, FInteger ncx, const
    FReal xbins[], FReal vmx = 0);
```

4.1.2 Profile histograms

See HBOOK manual, section 4.1.2.

```
inline void hbprof (FInteger id, const FCharacter *chtitl, FInteger ncx,
    FReal xlow, FReal xup, FReal ymin, FReal ymax, const FCharacter *chopt = " ");
```

```
inline void hbprof (FInteger id, const string& chtitl, FInteger ncx, FReal
    xlow, FReal xup, FReal ymin, FReal ymax, const string& chopt = " ");
```

Rounding

See HBOOK manual, section 4.1.3.

```
inline void hbinsz (const FCharacter *chopt);
```

```
inline void hbinsz (const string& chopt);
```

```
inline void hbinsz (bool opt);
```

This version of `hbinsz` takes a `bool` as argument, where `opt==true` corresponds to `chopt="YES"`, and `opt==false` corresponds to `chopt="NO"`.

Projections, slices, bands

See HBOOK manual, section 4.1.4.

```
inline void hbpro (FInteger id, FReal vmx = 0);
```

```
inline void hbprox (FInteger id, FReal vmx = 0);
```

```
inline void hbproy (FInteger id, FReal vmx = 0);
```

```
inline void hbandx (FInteger id, FReal ymi, FReal yma, FReal vmx = 0);
```

```
inline void hbandy (FInteger id, FReal xmi, FReal xma, FReal vmx = 0);
```

```
inline void hbslix (FInteger id, FInteger nsli, FReal vmx = 0);
```

```
inline void hbsliy (FInteger id, FInteger nsli, FReal vmx = 0);
```

Statistics

See HBOOK manual, section 4.1.5.

```
inline void hbarx (FInteger id);
```

```
inline void hbary (FInteger id);
```

```
inline void hbar2 (FInteger id);
```

Function representation

See HBOOK manual, section 4.1.6.

```
inline void hbfun1 (FInteger id, const FCharacter *chtitl, FInteger nx,
    FReal xmi, FReal xma, FReal fun (const FReal *));
```

```
inline void hbfun1 (FInteger id, const FCharacter *chtitl, FInteger nx,
    FReal xmi, FReal xma, FReal fun (const FReal&));
```

```
inline void hbfun1 (FInteger id, const string& chtitl, FInteger nx, FReal
    xmi, FReal xma, FReal fun (const FReal *));
```

```
inline void hbfun1 (FInteger id, const string& chtitl, FInteger nx, FReal
    xmi, FReal xma, FReal fun (const FReal&));
```

```
inline void hbfun2 (FInteger id, const FCharacter *chtitl, FInteger nx,
    FReal xmi, FReal xma, FInteger ny, FReal ymi, FReal yma, FReal fun (const
    FReal *, const FReal *));
```

```
inline void hbfun2 (FInteger id, const FCharacter *chtitl, FInteger nx,
    FReal xmi, FReal xma, FInteger ny, FReal ymi, FReal yma, FReal fun (const
    FReal&, const FReal&));
```

```
inline void hbfun2 (FInteger id, const string& chtitl, FInteger nx, FReal
    xmi, FReal xma, FInteger ny, FReal ymi, FReal yma, FReal fun (const
    FReal *, const FReal *));
```

```
inline void hbfun2 (FInteger id, const string& chtitl, FInteger nx, FReal
    xmi, FReal xma, FInteger ny, FReal ymi, FReal yma, FReal fun (const
    FReal&, const FReal&));
```

```
inline void hfunc (FInteger id, FReal fun (const FReal *));
```

```
inline void hfunc (FInteger id, FReal fun (const FReal&));
```

Reserve array in memory

See HBOOK manual, section 4.1.7.

```
inline FInteger harray (FInteger id, FInteger nwords, FInteger& loc);
```

All versions of `harray` return the value of `loc`.

```
inline FInteger harray (FInteger id, FInteger nwords);
```

Axis labels and histograms

See HBOOK manual, section 4.1.8.

```
template<int clablen>
void hlabel (FInteger id, FInteger nlab, FCharacter clab[][clablen], const
    FCharacter *chopt);
```

```
void hlabel (FInteger id, FInteger nlab, string clab[], const string& chopt);
```

4.3.2 Filling operations

See HBOOK manual, section 4.2.

Fast filling entries

See HBOOK manual, section 4.2.1.

```
inline void hf1 (FInteger id, FReal x, FReal weight);

inline void hf1e (FInteger id, FReal x, FReal weight, FReal errors);

inline void hf2 (FInteger id, FReal x, FReal y, FReal weight);

inline void hff1 (FInteger id, FInteger& nid, FReal x, FReal weight);

inline void hff2 (FInteger id, FInteger& nid, FReal x, FReal y, FReal weight);

inline void hfpak1 (FInteger id, FInteger& nid, const FReal v[], FInteger
    n);

inline void hfpak1 (FInteger id, const FReal v[], FInteger n);

inline void hipak1 (FInteger id, FInteger& nid, const FInteger iv[], FInteger
    n);

inline void hipak1 (FInteger id, const FInteger iv[], FInteger n);
```

Global filling

See HBOOK manual, section 4.2.2.

```
inline void hpak (FInteger id, const FReal conten[]);

inline void hpakad (FInteger id, const FReal conten[]);

inline void hpake (FInteger id, const FReal errors[]);
```

Filling histograms using character variables

See HBOOK manual, section 4.2.3.

```
inline void hfc1 (FInteger id, FInteger ibin, const FCharacter *clab, FReal
    w = 1, const FCharacter *chopt = " ");

inline void hfc1 (FInteger id, const FCharacter *clab, FReal w = 1, const
    FCharacter *chopt = " ");

inline void hfc1 (FInteger id, FInteger ibin, FReal w = 1, const FCharacter
    *chopt = " ");

inline void hfc1 (FInteger id, FInteger ibin, const string& clab, FReal
    w = 1, const string& chopt = " ");

inline void hfc1 (FInteger id, const string& clab, FReal w = 1, const string&
    chopt = " ");
```

```
inline void hfc1 (FInteger id, FInteger ibin, FReal w, const string& chopt);

inline void hfc2 (FInteger id, FInteger ibinx, const FCharacter *clabx,
    FInteger ibiny, const FCharacter *claby, FReal w = 1, const FCharacter
    *chopt = " ") ;

inline void hfc2 (FInteger id, const FCharacter *clabx, const FCharacter
    *claby, FReal w = 1, const FCharacter *chopt = " ");

inline void hfc2 (FInteger id, FInteger ibinx, FInteger ibiny, FReal w = 1,
    const FCharacter *chopt = " ");

inline void hfc2 (FInteger id, const FCharacter *clabx, FInteger ibiny,
    FReal w = 1, const FCharacter *chopt = " ");

inline void hfc2 (FInteger id, FInteger ibinx, const FCharacter *claby,
    FReal w = 1, const FCharacter *chopt = " ");

inline void hfc2 (FInteger id, FInteger ibinx, const string& clabx, FInteger
    ibiny, const string& claby, FReal w = 1, const string& chopt = " ");

inline void hfc2 (FInteger id, const string& clabx, const string& claby,
    FReal w = 1, const string& chopt = " ");

inline void hfc2 (FInteger id, const string& clabx, FInteger ibiny, FReal
    w = 1, const string& chopt = " ");

inline void hfc2 (FInteger id, FInteger ibinx, const string& claby, FReal
    w = 1, const string& chopt = " ");

inline void hfc2 (FInteger id, FInteger ibinx, FInteger ibiny, FReal w,
    const string& chopt);
```

4.3.3 Editing operations

See HBOOK manual, section 4.3.

Index and general title

See HBOOK manual, section 4.3.1.

```
inline void hindex ();

inline void htitle (const FCharacter *chgtit);

inline void htitle (const string& chgtit);
```

What to print (1-dimensional histogram)

See HBOOK manual, section 4.3.12.

```
inline void hidopt (FInteger id, const FCharacter *chopt = " ");

inline void hidopt (FInteger id, const string& chopt);

inline void hstaf (const FCharacter *chopt = " ");

inline void hstaf (const string& chopt);
```

Graphic choices (1-dimensional histogram)

See HBOOK manual, section 4.3.3.

```
inline void hpchar (const FCharacter *chopt, FCharacter charac);
```

```
inline void hpchar (const string& chopt, FCharacter charac);
```

```
inline void hbigbi (FInteger id, FInteger ncol);
```

Scale definition and normalization

See HBOOK manual, section 4.3.4.

```
inline void hmaxim (FInteger id, FReal fmax);
```

```
inline void hminim (FInteger id, FReal fmin);
```

```
inline void hcompa (const FInteger idvect[], FInteger n);
```

```
inline void hnorma (FInteger id, FReal xnorm);
```

```
inline void hscale (FInteger id, FReal factor);
```

Page control

See HBOOK manual, section 4.3.5.

```
inline void hsquez (const FCharacter *chopt);
```

```
inline void hsquez (const string& chopt);
```

```
inline void hsquez (bool opt);
```

```
inline void hpagsz (FInteger nlines);
```

selective editing

See HBOOK manual, section 4.3.6.

```
inline void hphist (FInteger id, const FCharacter *choice = " ", FInteger
    num = 0);
```

```
inline void hprot (FInteger id, const FCharacter *choice = " ", FInteger
    num = 0);
```

```
inline void hpscat (FInteger id);
```

```
inline void hptab (FInteger id);
```

```
inline void hphs (FInteger id);
```

```
inline void hphst (FInteger id);
```

Printing after system error recovery

See HBOOK manual, section 4.3.7.

```
inline void hponce ();
```

4.3.8 Changing logical unit numbers for output and message files

See HBOOK manual, section 4.3.8.

```
inline void houtpu (FInteger lout);
```

```
inline void hermes (FInteger lerr);
```

4.4 Accessing Information

See HBOOK manual chapter 5, page 95.

4.4.1 Testing if a histogram exists in memory

See HBOOK manual, section 5.1.

```
inline FLogical hexist (FInteger id);
```

The return value of `hexist` is the return the value of the HBOOK function `HEXIST`.

4.4.2 List of histograms

See HBOOK manual, section 5.2.

```
inline void hid1 (FInteger idvect[], FInteger &n);
```

```
inline void hid2 (FInteger idvect[], FInteger &n);
```

```
inline void hidall (FInteger idvect[], FInteger &n);
```

4.4.3 Number of entries

See HBOOK manual, section 5.3.

```
inline FInteger hnoent (FInteger id, FInteger& noent);
```

All versions of `hnoent` return the value of `noent`.

```
inline FInteger hnoent (FInteger id);
```

4.4.4 Histogram attributes contents

See HBOOK manual, section 5.4.

```
inline FInteger hkind (FInteger id, FInteger kind[], const FCharacter *chopt = " ");
```

All versions of `hkind` return the value of the first entry in the array `kind`.

```
inline FInteger hkind (FInteger id);
```

This version of `hkind` sets `chopt = " "`, so that only the first value of `kind` is returned, which is the function's return value.

```
inline FInteger hkind (FInteger id, FInteger kind[], const string& chopt);
```

4.4.5 Contents

See HBOOK manual, section 5.5.

```
inline void hunpak (FInteger id, FReal conten[], const FCharacter *choice = " ",
    FInteger num = 0);
```

```
inline void hunpak (FInteger id, FReal conten[], const string& choice, FInteger
    num = 0);
```

```
inline FReal hi (FInteger id, FInteger i);
```

The return value of `hi` is the return the value of the HBOOK function `HI`.

```
inline FReal hij (FInteger id, FInteger i, FInteger j);
```

The return value of `hij` is the return the value of the HBOOK function `HIJ`.

```
inline FReal hx (FInteger id, FReal x);
```

The return value of `hx` is the return the value of the HBOOK function `HX`.

```
inline FReal hxy (FInteger id, FReal x, FReal y);
```

The return value of `hxy` is the return the value of the HBOOK function `HXY`.

4.4.6 Errors

See HBOOK manual, section 5.6.

```
inline void hunpke (FInteger id, FReal conten[], const FCharacter *choice = " ",
    FInteger num = 0);
```

```
inline void hunpke (FInteger id, FReal conten[], const string& choice, FInteger
    num = 0);
```

```
inline FReal hie (FInteger id, FInteger i);
```

The return value of is the return the value of the HBOOK function .

```
inline FReal hije (FInteger id, FInteger i, FInteger j);
```

The return value of `hije` is the return the value of the HBOOK function `HIJE`.

```
inline FReal hxe (FInteger id, FReal x);
```

The return value of `hxe` is the return the value of the HBOOK function `HXE`.

```
inline FReal hxye (FInteger id, FReal x, FReal y);
```

The return value of `hxye` is the return the value of the HBOOK function `HXYE`.

4.4.7 Associated function

See HBOOK manual, section 5.7.

```
inline FReal hif (FInteger id, FInteger i);
```

The return value of `hif` is the return the value of the HBOOK function `HIF`.

4.4.8 Abscissa to channel number

See HBOOK manual, section 5.8.

```
inline FInteger hxi (FInteger id, FReal x, FInteger& i);
```

All versions of `hxi` return the value of the output variable `i`.

```
inline FInteger hxi (FInteger id, FReal x);
```

```
inline void hxyij (FInteger id, FReal x, FReal y, FInteger& i, FInteger&
    j);
```

```
inline FReal hix (FInteger id, FInteger i, FReal& x);
```

All versions of `hix` return the value of the output variable `x`.

```
inline FReal hix (FInteger id, FInteger i);
```

```
inline void hijxy (FInteger id, FInteger i, FInteger j, FReal& x, FReal&
    y);
```

4.4.9 Maximum and minimum

See HBOOK manual, section 5.9.

```
inline FReal hmax (FInteger id);
```

The return value of `hmax` is the return the value of the HBOOK function `HMAX`.

```
inline FReal hmin (FInteger id);
```

The return value of `hmin` is the return the value of the HBOOK function `HMIN`.

4.4.10 Rebinning

See HBOOK manual, section 5.10.

```
inline void hrebin (FInteger id, FReal x[], FReal y[], FReal ex[], FReal
    ey[], FInteger n, FInteger ifirst, FInteger ilast);
```

4.4.11 Integrated contents

See HBOOK manual, section 5.11.

```
inline FReal hsum (FInteger id);
```

The return value of `hsum` is the return the value of the HBOOK function `HSUM`.

4.4.12 Histogram definition

See HBOOK manual, section 5.12.

```
inline FReal hdump (FInteger id);
```

The return value of `hdump` is the return the value of the HBOOK function `HDUMP`.

```
template<int chtitllen>
```

```
void hgive (FInteger id, FCharacter chtitl [chtitllen], FInteger& nx, FReal&
    xmi, FReal& xma, FInteger& ny, FReal& ymi, FReal& yma, FInteger& nwt,
    FInteger& loc);
```

```
void hgive (FInteger id, string& chtitl, FInteger& nx, FReal& xmi, FReal&
    xma, FInteger& ny, FReal& ymi, FReal& yma, FInteger& nwt, FInteger&
    loc);
```

4.4.13 Statistics

See HBOOK manual, section 5.13.

```
inline void hstati (FInteger id, FInteger icase, const FCharacter *choice = " ",
    FInteger num = 0);
```

4.5 Operations on Histograms

See HBOOK manual chapter 6, page 103.

4.5.1 Arithmetic operations

See HBOOK manual, section 6.1.

```
inline void hopera (FInteger id1, const FCharacter *choper, FInteger id2,
    FInteger id3, FReal c1 = 1, FReal c2 = 1);
```

```
inline void hopera (FInteger id1, const string& choper, FInteger id2, FInteger
    id3, FReal c1 = 1, FReal c2 = 1);
```

4.5.2 Statistical differences between histograms

See HBOOK manual, section 6.2.

```
inline FReal hdiff (FInteger id1, FInteger id2, FReal& prob, const FCharacter
    *chopt);
```

All versions of `hdiff` return the value of the output variable `prob`.

```
inline FReal hdiff (FInteger id1, FInteger id2, const FCharacter *chopt);
```

```
inline FReal hdiff (FInteger id1, FInteger id2, FReal& prob, const string&
    chopt);
```

```
inline FReal hdiff (FInteger id1, FInteger id2, const string& chopt);
```

4.5.3 Bin by bin histogram comparisons

See HBOOK manual, section 6.3.

```
inline void hdiffb (FInteger id1, FInteger id2, FReal tol, FInteger nbin,
    const FCharacter *chopt, FInteger& nbad, FReal diff[]);
```

```
inline void hdiffb (FInteger id1, FInteger id2, FReal tol, FInteger nbin,
    const string& chopt, FInteger& nbad, FReal diff[]);
```

4.6 Fitting, parametrization and smoothing

See HBOOK manual chapter 7, page 113.

4.6.1 Fitting

See HBOOK manual, section 7.1.

One and two-dimensional distributions

See HBOOK manual, section 7.1.1.

```
inline void hfith (FInteger id, FReal fun(FReal *), const FCharacter *chopt,
    FInteger np, FReal param [], const FReal step [], const FReal pmin [],
    const FReal pmax [], FReal sigpar [], FReal& chi2);

inline void hfith (FInteger id, FReal fun(FReal&), const FCharacter *chopt,
    FInteger np, FReal param [], const FReal step [], const FReal pmin [],
    const FReal pmax [], FReal sigpar [], FReal& chi2);

inline void hfith (FInteger id, FReal fun(FReal *), const string& chopt,
    FInteger np, FReal param [], const FReal step [], const FReal pmin [],
    const FReal pmax [], FReal sigpar [], FReal& chi2);

inline void hfith (FInteger id, FReal fun(FReal& ), const string& chopt,
    FInteger np, FReal param [], const FReal step [], const FReal pmin [],
    const FReal pmax [], FReal sigpar [], FReal& chi2);
```

Fitting one-dimensional histograms with special functions

See HBOOK manual, section 7.1.2.

```
inline void hfitn (FInteger id, const FCharacter *chfun, const FCharacter
    *chopt, FInteger np, FReal param [], const FReal step [], const FReal
    pmin [], const FReal pmax [], FReal sigpar [], FReal& chi2);

inline void hfitn (FInteger id, const string& chfun, const string& chopt,
    FInteger np, FReal param [], const FReal step [], const FReal pmin [],
    const FReal pmax [], FReal sigpar [], FReal& chi2);
```

Fitting one or multi-dimensional arrays

See HBOOK manual, section 7.1.3.

```
inline void hfitv (FInteger n, FInteger ndim, FInteger nvar, const FReal
    x[], const FReal y[], const FReal ey[], FReal fun(FReal *), const FCharacter
    *chopt, FInteger np, FReal param [], const FReal step [], const FReal
    pmin [], const FReal pmax [], FReal sigpar [], FReal& chi2);

inline void hfitv (FInteger n, FInteger ndim, FInteger nvar, const FReal
    x[], const FReal y[], const FReal ey[], FReal fun(FReal&), const FCharacter
    *chopt, FInteger np, FReal param [], const FReal step [], const FReal
    pmin [], const FReal pmax [], FReal sigpar [], FReal& chi2);
```

```
inline void hfitv (FInteger n, FInteger ndim, FInteger nvar, const FReal
    x[], const FReal y[], const FReal ey[], FReal fun(FReal *), const string&
    chopt, FInteger np, FReal param [], const FReal step [], const FReal
    pmin [], const FReal pmax [], FReal sigpar [], FReal& chi2);
```

```
inline void hfitv (FInteger n, FInteger ndim, FInteger nvar, const FReal
    x[], const FReal y[], const FReal ey[], FReal fun(FReal& ), const string&
    chopt, FInteger np, FReal param [], const FReal step [], const FReal
    pmin [], const FReal pmax [], FReal sigpar [], FReal& chi2);
```

Naming the parameters of a fit

See HBOOK manual, section 7.1.4.

```
template<int chpnamlen>
void hfinam (FInteger id, const FCharacter chpnam[][chpnamlen], FInteger
    npar);
```

```
void hfinam (FInteger id, const string chpnam[], FInteger npar);
```

```
template <int fitnamlen>
void hgfit (FInteger id, FInteger& nfpar, FInteger& npfits, FReal& fitchi,
    FReal fitpar [], FReal fitsig [], FCharacter fitnam[][fitnamlen]);
```

```
void hgfit (FInteger id, FInteger& nfpar, FInteger& npfits, FReal& fitchi,
    FReal fitpar [], FReal fitsig [], string fitnam[]);
```

The user parametric function

See HBOOK manual, section 7.1.5.

```
inline void hderiv (FReal deriv[]);
```

4.6.2 Deprecated fitting routines

See HBOOK manual, section 7.3.

The following deprecated routines are not supported by this package:

```
hfitl
```

```
hfits
```

```
hfitn
```

```
hfit1
```

```
hfitex
```

```
hfitga
```

```
hfitpo
```

4.6.3 Parametrization

See HBOOK manual, section 7.4.

```
inline void hparam (FInteger id, FInteger ic, FReal r2min, const FInteger
    maxpow [], FInteger iterm [], FInteger& nco, FReal8 coeff []);
```

```
inline void hsetpr (const FCharacter *chname, FReal value);
```

```
inline void hsetpr (const string& chname, FReal value);
```

```
inline void hparmn (const FReal x [], const FReal y [], const FReal ey [],
    FInteger np, FInteger nvar, FInteger ic, FReal r2min, const FInteger
    maxpow [], FReal8 coeff [], FInteger iterm [], FInteger& nco);
```

4.6.4 Smoothing

See HBOOK manual, section 7.5.

```
inline FReal hsmoof (FInteger id, FInteger icase, FReal& chi2);
```

All versions of `hsmoof` return the value of the output variable `chi2`.

```
inline FReal hsmoof (FInteger id, FInteger icase = 0);
```

```
inline FReal hspli1 (FInteger id, FInteger ic, FInteger n, FInteger k, FReal&
    chi2);
```

All versions of `hspli1` return the value of the output variable `chi2`.

```
inline FReal hspli1 (FInteger id, FInteger ic, FInteger n, FInteger k);
```

```
inline void hspli2 (FInteger id, FInteger nx, FInteger ny, FInteger kx,
    FInteger ky);
```

```
inline FReal hspfun (FInteger id, FReal x, FInteger n, FInteger k);
```

The return value of `hspfun` is the return the value of the HBOOK function `HSPFUN`.

```
inline FInteger hquad (FInteger id, const FCharacter *chopt, FInteger mode,
    FReal sensit, FReal smooth, FInteger& nsig, FReal& chisq, FInteger&
    ndf, FReal& fmin, FReal& fmax, FInteger& ierr);
```

All versions of `hquad` return the value of the errorflag `ierr`.

```
inline FInteger hquad (FInteger id, const FCharacter *chopt, FInteger mode,
    FReal sensit, FReal smooth, FInteger& nsig, FReal& chisq, FInteger&
    ndf, FReal& fmin, FReal& fmax);
```

```
inline FInteger hquad (FInteger id, const string& chopt, FInteger mode,
    FReal sensit, FReal smooth, FInteger& nsig, FReal& chisq, FInteger&
    ndf, FReal& fmin, FReal& fmax, FInteger& ierr);
```

```
inline FInteger hquad (FInteger id, const string& chopt, FInteger mode,
    FReal sensit, FReal smooth, FInteger& nsig, FReal& chisq, FInteger&
    ndf, FReal& fmin, FReal& fmax);
```

4.6.5 Random number generation

See HBOOK manual, section 7.6.

`inline FReal hrndm1 (FInteger id);`

The return value of `hrndm1` is the return the value of the HBOOK function `HRNDM1`.

`inline void hrndm2 (FInteger id, FReal& rx, FReal& ry);`

4.6.6 Fitting with Monte Carlo statistics

See HBOOK manual, section 7.7.

```
inline void hmcmll (FInteger idd, const FInteger idm[], const FInteger idw[],
    FInteger nsrc, const FCharacter *chopt, const FInteger ifix[], const
    FReal frc[], const FReal flim[][2], const FReal start[], const FReal
    step[], FReal up, FReal par[], FReal dpar[]);
```

```
inline void hmcmll (FInteger idd, const FInteger idm[], const FInteger idw[],
    FInteger nsrc, const string& chopt, const FInteger ifix[], const FReal
    frc[], const FReal flim[][2], const FReal start[], const FReal step[],
    FReal up, FReal par[], FReal dpar[]);
```

```
inline FInteger hmcini (FInteger iddata, const FInteger idmc[], const FInteger
    idwt[], FInteger nsrc, const FCharacter *chopt, FInteger& ierr);
```

All versions of `hmcini` return the value of the errorflag `ierr`.

```
inline FInteger hmcini (FInteger iddata, const FInteger idmc[], const FInteger
    idwt[], FInteger nsrc, const FCharacter *chopt = " ") ;
```

```
inline FInteger hmcini (FInteger iddata, const FInteger idmc[], const FInteger
    idwt[], FInteger nsrc, const string& chopt, FInteger& ierr);
```

```
inline FInteger hmcini (FInteger iddata, const FInteger idmc[], const FInteger
    idwt[], FInteger nsrc, const string& chopt);
```

`inline FReal8 hmclnl (const FReal frac[]);`

The return value of `hmclnl` is the return the value of the HBOOK function `HMCLNL`.

4.7 Memory management and input/output routines

See HBOOK manual chapter 8, page 161.

4.7.1 Memory size control

See HBOOK manual, section 8.2.

`inline void hlimit (FInteger npaw);`

`inline FInteger hlocat (FInteger id, FInteger &loc);`

All versions of `hlocat` return the value of the output variable `loc`.

`inline FInteger hlocat (FInteger id);`

4.7.2 Directories

See HBOOK manual, section 8.3.

```
inline void hmdir (const FCharacter *chpath, const FCharacter *chopt = " ");
```

```
inline void hmdir (const string& chpath, const string& chopt = " ");
```

```
inline void hcdir (const FCharacter *chpath, const FCharacter *chopt = " ");
```

```
inline void hcdir (const string& chpath, const string& chopt = " ");
```

```
inline void hldir (const FCharacter *chpath, const FCharacter *chopt = " ");
```

```
inline void hldir (const string& chpath, const string& chopt = " ");
```

```
inline void hddir (const FCharacter *chpath = " ");
```

```
inline void hddir (const string& chpath);
```

```
inline void hpdir (const FCharacter *chpath, const FCharacter *chopt = " ");
```

```
inline void hpdir (const string& chpath, const string& chopt = " ");
```

```
template<int chtypelen, int chtitllen>
FInteger hlnext (FInteger& idh, FCharacter chtype[chtypelen], FCharacter
    chtitl[chtitllen], const FCharacter *chopt = " ");
```

All versions of `hlnext` return the value of the output variable `idh`.

```
FInteger hlnext (FInteger& idh, string chtype, string chtitl, const FCharacter
    *chopt = " ");
```

```
template<int chdirlen>
FInteger hrdir (FInteger maxdir, FCharacter chdir[][chdirlen], FInteger
    &ndir);
```

All versions of `hrdir` return the value of the output variable `ndir`.

```
template<int chdirlen>
FInteger hrdir (FInteger maxdir, FCharacter chdir[][chdirlen]);
```

```
FInteger hrdir (FInteger maxdir, string chdir[], FInteger &ndir);
```

```
FInteger hrdir (FInteger maxdir, string chdir[]);
```

4.7.3 Input/output routines

See HBOOK manual, section 8.4.

```
inline void hrput (FInteger id = 0, const FCharacter *chfile = " ", const
    FCharacter *chopt = "N");
```

```
inline void hrput (FInteger id, const string& chfile, const string& chopt = "N");
```

```
inline void hrget (FInteger id = 0, const FCharacter *chfile = " ", const
    FCharacter *chopt = "A");
```

```
inline void hrget (FInteger id, const string& chfile, const string& chopt = "A");
```

```
inline FInteger hropen (FInteger lun, const FCharacter *chtop, const FCharacter
    *chfile, const FCharacter *chopt, FInteger& lrec, FInteger& istat);
```

All versions of `hropen` return the value of the errorflag `istat`.

```
inline FInteger hropen (FInteger lun, const FCharacter *chtop, const FCharacter
    *chfile, const FCharacter *chopt, FInteger& lrec);
    /*
    // The following two overloaded functions do not compile under aCC

inline int hropen (FInteger lun, const FCharacter *chtop, const FCharacter
    *chfile, const FCharacter *chopt, const FInteger lrec, FInteger& istat);

inline int hropen (FInteger lun, const FCharacter *chtop, const FCharacter
    *chfile, const FCharacter *chopt = " ", const FInteger lrec = 0);
    */

inline int hropen (FInteger lun, const string& chtop, const string& chfile,
    const string& chopt, FInteger& lrec, FInteger& istat);

inline int hropen (FInteger lun, const string& chtop, const string& chfile,
    const string& chopt, FInteger& lrec);
    /*
    // The following two overloaded functions do not compile under aCC

inline int hropen (FInteger lun, const string& chtop, const string& chfile,
    const string& chopt, const FInteger lrec, FInteger& istat);

inline int hropen (FInteger lun, const string& chtop, const string& chfile,
    const string& chopt = " ", const FInteger lrec = 0);
    */

inline void hrfile (FInteger lun, const FCharacter *chtop, const FCharacter
    *chopt);

inline void hrfile (FInteger lun, const string& chtop, const string& chopt);

inline FInteger hrout (FInteger id, FInteger& icycle, const FCharacter *chopt = " ");
```

All versions of `hrout` return the value of the output variable `icycle`.

```
inline FInteger hrout (FInteger id, const FCharacter *chopt = " ");
```

This version of `hrout` sets the initial value of the variable `icycle` to zero.

```
inline FInteger hrout (FInteger id, const string& chopt);

inline void hrin (FInteger id, FInteger icycle, FInteger ioffset = 0);

template<int chfinlen>
inline void hmerge (FInteger nfiles, const FCharacter chfin [][chfinlen],
    const FCharacter *chfout);

void hmerge (FInteger nfiles, const string chfin [], const string& chfout);

inline void hmergin ();
```

```
inline void hscr (FInteger id = 0, FInteger icycle = 0, const FCharacter
    *chopt = " ");
```

```
inline void hscr (FInteger id, FInteger icycle, const string& chopt);
```

```
inline void hrend (const FCharacter *chtop = " ");
```

```
inline void hrend (const string& chtop);
```

References

[CER95] CERN Computing and Networks Division (1995): HBOOK reference manual, version 4.24. — *CERN program library long writeup Y250.*

[Lip96] Stanley B. Lippman (1996): C++ gems. — *xxiii+601 pp., New York, NY (SIGS Books).*

[Sch96] Jerry Schwarz (1996): Initializing static variables in C++ libraries. — *In: [Lip96], p. 237–241.*

[Str97] Bjarne Stroustrup (1997): The C++ programming language (3rd ed.). — *910 pp., Reading, MA (Addison Wesley).*