```c
// Created by hengxin on 2024/12/04.

int main() {
  char **argv;

  int *names[10];

  int (*musician_score_table)[10];

  char *StrCpyStd(char *dest, const char *src);

  int (*comp)(const void *left, const void *right);

  // see https://en.cppreference.com/w/c/program/atexit
  int atexit(void (*func)(void));

  // see https://en.cppreference.com/w/c/program/signal
  void (*signal(int sig, void (*handler)(int)))(int);

  // typedef void (*sighandler_t)(int);
  // sighandler_t Signal(int sig, sighandler_t handler);

  char (*(*func(int num, char *str))[3])(void);

  char (*(*arr[3])(void))[5];

  // Refer to https://cdecl.org/ for more practice.
  // See https://c-faq.com/decl/spiral.anderson.html for secrets!!!
}
```

```c
// Created by hengxin on 2024/12/04.

#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <string.h>

// (since C11)
// _Generic ( controlling-expression , association-list )
// See Section 9.7 of the textbook
#define Print(x, y) _Generic((x), \
    int *: PrintInts, \
    const char **: PrintStrs \
    )((x), (y))

typedef int (*CompareFunction)(const void *, const void
 *);
typedef int CompFunc(const void *, const void *);

int CompareInts(const void *left, const void *right);
int CompareStrs(const void *left, const void *right);
int CompareStrsWrong(const void *left, const void *right);

void PrintInts(const int *integers, size_t len);
void PrintStrs(const char *str[], size_t len);

int main() {
  int integers[] = {-2, 99, 0, -743, 2, INT_MIN, 4};
  int size_of_integers = sizeof integers / sizeof *
integers;

  /**
   * void qsort( void *ptr, size_t count, size_t size,
               int (*comp)(const void *, const void *) );
   * typedef int _Cmpfun(const void *, const void *);
   * void qsort( void *ptr, size_t count, size_t size,
_Cmpfun *comp);
   */
  int (*comp)(const void *, const void *) = CompareInts;

  // CompareFunction comp1 = CompareInts;
  // CompFunc *comp2 = CompareInts;

  // you should not do this!!!
  // printf("sizeof comp : %zu\n", sizeof comp);
  printf("comp : %p\n", comp);
  printf("*comp : %p\n", *comp);
  printf("CompareInts : %p\n", CompareInts);
  printf("&CompareInts : %p\n", &CompareInts);

  qsort(integers, size_of_integers, sizeof *integers, comp
);
  // PrintInts(integers, size_of_integers);
  Print(integers, size_of_integers);

  // Call functions indirectly via function pointers.
  int a = 10;
  int b = 20;
  printf("%d %s %d\n", a, comp(&a, &b) > 0 ? ">" : "<=", b
);

  const char *names[] = {
      "Luo Dayou",
      "Cui Jian",
      "Dou Wei",
      "Zhang Chu",
      "Wan Qing",
      "Li Zhi",
      "Yao",
      "ZuoXiao",
      "ErShou Rose",
      "Hu Mage",
  };
  size_t size_of_names = sizeof names / sizeof *names;

  comp = CompareStrs;
  // qsort(names, size_of_names,
  //       sizeof *names, comp);
  // PrintStrs(names, size_of_names);

  // comp = CompareStrsWrong;
  comp = CompareStrs;
  qsort(names, size_of_names,
        sizeof *names, comp);
  // PrintStrs(names, size_of_names);
  Print(names, size_of_names);
}

```

```c
int CompareInts(const void *left, const void *right) {
  int int_left = *(const int *) left;
  int int_right = *(const int *) right;

  if (int_left < int_right) {
    return -1;
  }

  if (int_left > int_right) {
    return 1;
  }

  return 0;

  // return (int_left > int_right) - (int_left < int_right);
  // return int_left - int_right; // erroneous shortcut (fails if INT_MIN is present)
}

int CompareStrs(const void *left, const void *right) {
  const char *const *pp1 = left;
  const char *const *pp2 = right;
  return strcmp(*pp1, *pp2);
}

// Why keep the original order???
// What are compared???
int CompareStrsWrong(const void *left, const void *right) {
  const char *pp1 = left;
  const char *pp2 = right;
  return strcmp(pp1, pp2);
}

void PrintInts(const int *integers, size_t len) {
  printf("\n");
  for (int i = 0; i < len; i++) {
    printf("%d ", integers[i]);
  }
  printf("\n");
}

void PrintStrs(const char *str[], size_t len) {
  printf("\n");
  for (int i = 0; i < len; i++) {
    printf("%s\n", str[i]);
  }
  printf("\n");
}
```

```c
// Created by hfwei on 2024/12/04.
// See https://en.cppreference.com/w/c/program/atexit

#include <stdlib.h>
#include <stdio.h>

void f1(void) {
  puts("f1");
}

void f2(void) {
  puts("f2");
}

int main(void) {
  if (!atexit(f1) && !atexit(f2) && !atexit(f2)) {
    return EXIT_SUCCESS;
  }

  // atexit registration failed
  return EXIT_FAILURE;

}   // <- if registration was successful calls f2, f2, f1
```

```c
// Created by hfwei on 2024/12/04.

#include <stdio.h>
#include <signal.h>

void SIGSEGV_Handler(int sig) {
  printf("SIGSEGV %d is caught.\n", sig);
}

int main(void) {
  signal(SIGSEGV, SIGSEGV_Handler);
//  raise(SIGSEGV);

  int *p = NULL;
  *p = 0;

  return 0;
}
```

```c
// Created by hfwei on 2024/12/04.

#include <stdio.h>

// See https://elixir.bootlin.com/linux/latest/source/include/linux/types.h#L245
typedef int (*cmp_func_t)(const void *a, const void *b);

// See https://elixir.bootlin.com/linux/latest/source/include/linux/bsearch.h#L8
void *bsearch(const void *key, const void *base,
              size_t num, size_t size, cmp_func_t cmp);

int main(void) {

  return 0;
}

void *bsearch(const void *key, const void *base, size_t num, size_t size, cmp_func_t cmp) {
  const char *pivot;
  int result;

  while (num > 0) {
    pivot = base + (num >> 1) * size;
    result = cmp(key, pivot);

    if (result == 0) {
      return (void *) pivot;
    }

    if (result > 0) {
      base = pivot + size;
      num--;
    }

    num >>= 1;
  }

  return NULL;
}
```

```
# `11-function-pointers`

## `integrate.c`

## `sort.c`

## `bsearch-gnuc.c`

## `decl.c`
```

```c
// Created by hfwei on 2024/12/04.
// A nice function pointer example on Riemann integration:
// https://en.wikipedia.org/wiki/Function_pointer

#include <stdio.h>
#include <math.h>

#define NUM_OF_PARTITIONS 1000000

double Integrate(double low, double high, double (*func)(
double));

double Square(double x);

int main() {
  double low = 0.0;
  double high = 1.0;
  double integration = 0.0;

  // gcc -pedantic (invalid application of sizeof to a
function type)
  // See "Function to pointer conversion" (https://en.
cppreference.com/w/c/language/conversion)
  // See also https://en.cppreference.com/w/c/language/
sizeof
  printf("sizeof sin: %zu\n", sizeof sin);
  printf("sizeof &sin: %zu\n", sizeof &sin);

  integration = Integrate(low, high, sin);
  printf("sin(x) from %f to %f is %f\n", low, high,
integration);

  integration = Integrate(low, high, cos);
  printf("cos(x) from %f to %f is %f\n", low, high,
integration);

  integration = Integrate(low, high, Square);
  printf("Square(x) from %f to %f is %f\n", low, high,
integration);

  double (*funcs[])(double) = {sin, cos, Square};

  int len = sizeof(funcs) / sizeof(*funcs);
  for (int i = 0; i < len; ++i) {
    printf("integration: %f\n", Integrate(low, high, funcs[i]));
  }

  return 0;
}

double Integrate(double low, double high, double (*func)(double)) {
  double interval = (high - low) / NUM_OF_PARTITIONS;

  double sum = 0.0;
  for (int i = 0; i < NUM_OF_PARTITIONS; i++) {
    double xi = low + interval * i;
    double yi = func(xi);
    sum += yi * interval;
  }

  return sum;
}

double Square(double x) {
  return x * x;
}
```

```c
// Created by hfwei on 2024/12/04.
// Question: What if char key_name[] = "Zhang Chu"?

#include <stdio.h>
#include <string.h>
#include <stdbool.h>

// See https://codebrowser.dev/glibc/glibc/stdlib/stdlib.h.html#__compar_fn_t
// The first is a pointer to the key for the search,
// and the second is a pointer to the array element to be compared with the key.
typedef int (*__compar_fn_t)(const void *, const void *);

// See https://codebrowser.dev/glibc/glibc/bits/stdlib-bsearch.h.html#19
void *bsearch(const void *__key, const void *__base,
              size_t __nmemb, size_t __size,
              __compar_fn_t __compar);
void *bsearch_leftmost(const void *__key, const void *__base,
                       size_t __nmemb, size_t __size,
                       __compar_fn_t __compar);

int CompareStrs(const void *left, const void *right);
int CompareStrsCI(const void *left, const void *right);
int CompareStrsAddress(const void *left, const void *right);

// int (*GetCompareFunction(bool case_sensitive))(const void *, const void *);
__compar_fn_t GetCompareFunction(bool case_sensitive) {
  return case_sensitive ? &CompareStrs : &CompareStrsCI;
}

const char *names[] = {
    "Cui Jian",
    "Dou Wei",
    "ErShou Rose",
    "Hu Mage",
    "Li Zhi",
    "Luo Dayou",
    "Wan Qing",
    "Yao",
    "Zhang Chu",
    "Zhang Chu",
    "Zhang Chu",
    "Zhang Chu",
    "ZuoXiao",
};

int main(void) {
  char *key_name = "Zhang Chu";

  // char **name_ptr = bsearch(&key_name, names,
  //                           sizeof names / sizeof *
names,
  //                           sizeof *names,
  //                           CompareStrs);

  // char **name_ptr = bsearch(&key_name, names,
  //                           sizeof names / sizeof *
names,
  //                           sizeof *names,
  //                           CompareStrsAddress);

  char **name_ptr = bsearch_leftmost(&key_name, names,
                                     sizeof names / sizeof
 *names,
                                     sizeof *names,
                                     CompareStrsAddress);

  if (name_ptr != NULL) {
    printf("Found %s at index %lld.\n",
           *name_ptr, name_ptr - (char **) names);
  } else {
    printf("Could not find %s.\n", key_name);
  }

  char *key_name_ci = "zhang chu";

  char **name_ci_ptr = bsearch(&key_name_ci, names,
                               sizeof names / sizeof *
names,
                               sizeof *names,
                               GetCompareFunction(false));
  if (name_ci_ptr != NULL) {
    printf("Found %s at index %lld.\n",
           *name_ci_ptr,
           name_ci_ptr - (char **) names);
  } else {
    printf("Could not find %s.\n", key_name_ci);
  }

  return 0;
}

// Visualization: https://pythontutor.com/render.html#code=//%0A//%20Created%20by%20hfwei%20on%202023/12/13.%0A//%20Question%3A%20What%20if%20char%20key_name%5B%5D%20%3D%20%22Zhang%20Chu%22%3F%0A//%0A%0A%23include%20%3Cstdio.h%3E%0A%23include%20%3Cstring.h%3E%0A%0A//%20See%20https%3A//codebrowser.dev/glibc/glibc/stdlib/stdlib.h.html%23__compar_fn_t%0A//%20The%20first%20is%20a%20pointer%20to%20the%20key%20for%20the%20search,%0A//%20and%20the%20second%20is%20a%20pointer%20to%20the%20array%20element%20to%20be%20compared%20with%20the%20key.%0Atypedef%20int%20%28*__compar_fn_t%29%28const%20void%20*,%20const%20void%20*%29%3B%0A%0A//%20See%20https%3A//codebrowser.dev/glibc/glibc/bits/stdlib-bsearch.h.html%2319%0Avoid%20*bsearch%28const%20void%20*__key,%20const%20void%20*__base,%20size_t%20__nmemb,%20size_t%20__size,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20__compar_fn_t%20__compar%29%3B%0A%0Aint%20CompareStrs%28const%20void%20*left,%20const%20void%20*right%29%3B%0Aint%20CompareStrsAddress%28const%20char%20*left,%20const%20char%20*right%29%3B%0A%0Aconst%20char%20*names%5B%5D%20%3D%20%7B%0A%20%20%20%20%22Cui%20Jian%22,%0A%20%20%20%20%22Dou%20Wei%22,%0A%20%20%20%20%22ErShou%20Rose%22,%0A%20%20%20%20%22Hu%20Mage%22,%0A%20%20%20%20%22Li%20Zhi%22,%0A%20%20%20%20%22Luo%20Dayou%22,%0A%20%20%20%20%22Wan%20Qing%22,%0A%20%20%20%20%22Yao%22,%0A%20%20%20%20%22Zhang%20Chu%22,%0A%20%20%20%20%22ZuoXiao%22,%0A%7D%3B%0A%0Aint%20main%28void%29%20%7B%0A%20%20char%20*key_name%20%3D%20%22Zhang%20Chu%22%3B%0A%0A%20%20//%20char%20**name_ptr%20%3D%20bsearch%28%26key_name,%20names,%0A%20%20//%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20sizeof%20names%20/%20sizeof%20*names,%0A%20%20//%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20sizeof%20*names,%0A%20%20//%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%28__compar_fn_t%29%20strcmp%29%3B%20//%
```

```
20CompareStrsAddress%0A%0A%20%20char%20**name_ptr%20%3D%
20bsearch%28%26key_name,%20names,%0A%20%20%20%20%20%20%20
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20
%20%20sizeof%20names%20/%20sizeof%20*names,%0A%20%20%20%
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%
20%20%20%20%20%20sizeof%20*names,%0A%20%20%20%20%20%20%20
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20
%20%20CompareStrs%29%3B%0A%0A%20%20if%20%28*name_ptr%20!%
3D%20NULL%29%20%7B%0A%20%20%20%20printf%28%22Found%20%25s
.%5Cn%22,%20*name_ptr%29%3B%0A%20%20%7D%20else%20%7B%0A%
20%20%20%20printf%28%22Could%20not%20find%20%25s.%5Cn%22
,%20key_name%29%3B%0A%20%20%7D%0A%0A%20%20return%200%3B%
0A%7D%0A%0Aint%20CompareStrs%28const%20void%20*left,%
20const%20void%20*right%29%20%7B%0A%20%20char%20*const%20
*pp1%20%3D%20left%3B%0A%20%20char%20*const%20*pp2%20%3D%
20right%3B%0A%20%20return%20strcmp%28*pp1,%20*pp2%29%3B%
0A%7D%0A%0A//%20What%20is%20the%20advantage%20of%20this%
20version%3F%20%28performance%3F%3F%3F%29%0A//%20What%
20is%20the%20disadvantage%20of%20this%20version%3F%20%
28not%20flexible%3F%3F%3F%29%0Aint%20CompareStrsAddress%
28const%20char%20*left,%20const%20char%20*right%29%20%7B%
0A%20%20return%20strcmp%28left,%20right%29%3B%0A%7D%0A%
0Avoid%20*bsearch%28const%20void%20*__key,%20const%20void
%20*__base,%20size_t%20__nmemb,%20size_t%20__size,%0A%20%
20%20%20%20%20%20%20%20%20%20%20%20%20__compar_fn_t%
20__compar%29%20%7B%0A%20%20size_t%20__l,%20__u,%20__idx%
3B%0A%20%20const%20void%20*__p%3B%0A%20%20int%
20__comparison%3B%0A%20%20__l%20%3D%200%3B%0A%20%20__u%20
%3D%20__nmemb%3B%0A%20%20while%20%28__l%20%3C%20__u%29%20
%7B%0A%20%20%20%20__idx%20%3D%20%28__l%20%2B%20__u%29%20
/%202%3B%0A%20%20%20%20__p%20%3D%20%28const%20void%20*%29
%20%28%28%28const%20char%20*%29%20__base%29%20%2B%20%
28__idx%20*%20__size%29%29%3B%0A%20%20%20%20__comparison%
20%3D%20%28*__compar%29%28__key,%20__p%29%3B%0A%20%20%20%
20if%20%28__comparison%20%3C%200%29%20%7B%0A%20%20%20%20%
20%20__u%20%3D%20__idx%3B%0A%20%20%20%20%7D%20else%20if%
20%28__comparison%20%3E%200%29%20%7B%0A%20%20%20%20%20%
20__l%20%3D%20__idx%20%2B%201%3B%0A%20%20%20%20%7D%20else
%20%7B%0A%20%20%20%20%20%20return%20%28void%20*%29%20__p%
3B%0A%20%20%20%20%7D%0A%20%20%7D%0A%0A%20%20return%20NULL
%3B%0A%7D&cppShowMemAddrs=true&cumulative=true&curInstr=
14&heapPrimitives=nevernest&mode=display&origin=opt-
frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&
textReferences=false
```

```c
int CompareStrs(const void *left, const void *right) {
  char *const *pp1 = left;
  char *const *pp2 = right;
  return strcmp(*pp1, *pp2);
}

int CompareStrsCI(const void *left, const void *right) {
  const char *const *pp1 = left;
  const char *const *pp2 = right;
  // see https://www.ibm.com/docs/en/zos/2.4.0?topic=functions-strcasecmp-case-insensitive-string-comparison
  return strcasecmp(*pp1, *pp2);
}

// What is the advantage of this version? (performance ???)
// What is the disadvantage of this version? (not flexible???)
// Visualization: https://pythontutor.com/render.html#code=//%0A//%20Created%20by%20hfwei%20on%202023/12/13.%0A//%20Question%3A%20What%20if%20char%20key_name%5B%5D%20%3D%20%22Zhang%20Chu%22%3F%0A//%0A%0A%23include%20%3Cstdio.h%3E%0A%23include%20%3Cstring.h%3E%0A%0A//%20See%20https%3A//codebrowser.dev/glibc/glibc/stdlib/stdlib.h.html%23__compar_fn_t%0A//%20The%20first%20is%20a%20pointer%20to%20the%20key%20for%20the%20search,%0A//%20and%20the%20second%20is%20a%20pointer%20to%20the%20array%20element%20to%20be%20compared%20with%20the%20key.%0Atypedef%20int%20%28*__compar_fn_t%29%28const%20void%20*,%20const%20void%20*%29%3B%0A%0A//%20See%20https%3A//codebrowser.dev/glibc/glibc/bits/stdlib-bsearch.h.html%2319%0Avoid%20*bsearch%28const%20void%20*__key,%20const%20void%20*__base,%20size_t%20__nmemb,%20size_t%20__size,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20__compar_fn_t%20__compar%29%3B%0A%0Aint%20CompareStrs%28const%20void%20*left,%20const%20void%20*right%29%3B%0Aint%20CompareStrsAddress%28const%20char%20*left,%20const%20char%20*right%29%3B%0A%0Aconst%20char%20*names%5B%5D%20%3D%20%7B%0A%20%20%20%20%22Cui%20Jian%22,%0A%20%20%20%20%22Dou%20Wei%22,%0A%20%20%20%20%22ErShou%20Rose%22,%0A%20%20%20%20%22Hu%20Mage%22,%0A%20%20%20%20%22Li%20Zhi%22,%0A%20%20%20%20%22Luo%20Dayou%22,%0A%20%20%20%20%22Wan%20Qing%22,%0A%20%20%20%20%22Yao%22,%0A%20%20%20%20%22Zhang%20Chu%22,%0A%20%20%20%20%22ZuoXiao%22,%0A%7D%3B%0A%0Aint%20main%28void%29%20%7B%0A%20%
```

```
20char%20*key_name%20%3D%20%22Zhang%20Chu%22%3B%0A%0A%20%
20//%20char%20**name_ptr%20%3D%20bsearch%28%26key_name,%
20names,%0A%20%20//%20%20%20%20%20%20%20%20%20%20%20%20%
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20sizeof%
20names%20/%20sizeof%20*names,%0A%20%20//%20%20%20%20%20%
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%
20%20%20sizeof%20*names,%0A%20%20//%20%20%20%20%20%20%20%
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%
20%28__compar_fn_t%29%20strcmp%29%3B%20//%
20CompareStrsAddress%0A%0A%20%20char%20**name_ptr%20%3D%
20bsearch%28%26key_name,%20names,%0A%20%20%20%20%20%20%20
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20
%20%20sizeof%20names%20/%20sizeof%20*names,%0A%20%20%20%
20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%
20%20%20%20%20%20sizeof%20*names,%0A%20%20%20%20%20%20%20
%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20
%20%20CompareStrsAddress%29%3B%0A%0A%20%20if%20%28*
name_ptr%20!%3D%20NULL%29%20%7B%0A%20%20%20%20printf%28%
22Found%20%25s.%5Cn%22,%20*name_ptr%29%3B%0A%20%20%7D%
20else%20%7B%0A%20%20%20%20printf%28%22Could%20not%20find
%20%25s.%5Cn%22,%20key_name%29%3B%0A%20%20%7D%0A%0A%20%
20return%200%3B%0A%7D%0A%0Aint%20CompareStrs%28const%
20void%20*left,%20const%20void%20*right%29%20%7B%0A%20%
20char%20*const%20*pp1%20%3D%20left%3B%0A%20%20char%20*
const%20*pp2%20%3D%20right%3B%0A%20%20return%20strcmp%28*
pp1,%20*pp2%29%3B%0A%7D%0A%0A//%20What%20is%20the%
20advantage%20of%20this%20version%3F%20%28performance%3F%
3F%3F%29%0A//%20What%20is%20the%20disadvantage%20of%
20this%20version%3F%20%28not%20flexible%3F%3F%3F%29%0Aint
%20CompareStrsAddress%28const%20char%20*left,%20const%
20char%20*right%29%20%7B%0A%20%20return%20strcmp%28left,%
20right%29%3B%0A%7D%0A%0Avoid%20*bsearch%28const%20void%
20*__key,%20const%20void%20*__base,%20size_t%20__nmemb,%
20size_t%20__size,%0A%20%20%20%20%20%20%20%20%20%20%20%20
%20%20__compar_fn_t%20__compar%29%20%7B%0A%20%20size_t%
20__l,%20__u,%20__idx%3B%0A%20%20const%20void%20*__p%3B%
0A%20%20int%20__comparison%3B%0A%20%20__l%20%3D%200%3B%0A
%20%20__u%20%3D%20__nmemb%3B%0A%20%20while%20%28__l%20%3C
%20__u%29%20%7B%0A%20%20%20%20__idx%20%3D%20%28__l%20%2B%
20__u%29%20/%202%3B%0A%20%20%20%20__p%20%3D%20%28const%
20void%20*%29%20%28%28%28const%20char%20*%29%20__base%29%
20%2B%20%28__idx%20*%20__size%29%29%3B%0A%20%20%20%
20__comparison%20%3D%20%28*__compar%29%28__key,%20__p%29%
3B%0A%20%20%20%20if%20%28__comparison%20%3C%200%29%20%7B%
```

```c
0A%20%20%20%20%20%20__u%20%3D%20__idx%3B%0A%20%20%20%20%7D%20else%20if%20%28__comparison%20%3E%200%29%20%7B%0A%20%20%20%20%20%20__l%20%3D%20__idx%20%2B%201%3B%0A%20%20%20%20%7D%20else%20%7B%0A%20%20%20%20%20%20return%20%28void%20*%29%20__p%3B%0A%20%20%20%20%7D%0A%20%20%7D%0A%0A%20%20return%20NULL%3B%0A%7D&cppShowMemAddrs=true&cumulative=true&curInstr=30&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=c_gcc9.3.0&rawInputLstJSON=%5B%5D&textReferences=false
int CompareStrsAddress(const void *left, const void *right) {
  const char *pp1 = left;
  const char *pp2 = right;
  return strcmp(pp1, pp2);
}

void *bsearch(const void *__key, const void *__base, size_t __nmemb, size_t __size,
              __compar_fn_t __compar) {
  size_t __l, __u, __idx;
  const void *__p;
  int __comparison;
  __l = 0;
  __u = __nmemb;
  while (__l < __u) {
    __idx = (__l + __u) / 2;
    __p = (const void *) (((const char *) __base) + (__idx * __size));
    __comparison = (*__compar)(__key, __p);
    if (__comparison < 0) {
      __u = __idx;
    } else if (__comparison > 0) {
      __l = __idx + 1;
    } else {
      return (void *) __p;
    }
  }

  return NULL;
}

void *bsearch_leftmost(const void *__key, const void *__base,
                       size_t __nmemb, size_t __size,
                            __compar_fn_t __compar) {
  size_t __l, __u, __idx;
  const void *__p;
  int __comparison;

  __l = 0;
  __u = __nmemb;
  // added by ant
  void *__index = NULL;

  while (__l < __u) {
    __idx = (__l + __u) / 2;
    __p = (const void *) (((const char *) __base) + (
 __idx * __size));
    __comparison = (*__compar)(__key, __p);
    if (__comparison < 0) {
      __u = __idx;
    } else if (__comparison > 0) {
      __l = __idx + 1;
    } else {
      // added by ant
      __index = (void *) __p;
      __u = __idx - 1;
    }
  }

  // added by ant
  return __index;
}
```

```
add_executable(integrate integrate.c)

add_executable(sort sort.c)

add_executable(bsearch bsearch.c)
add_executable(bsearch-gnuc bsearch-gnuc.c)

add_executable(11-decl decl.c)
add_executable(atexit atexit.c)
add_executable(signal signal.c)
```