```
1  # 7-data-types
2
3  ## `int-limits.c`
4
5  ## `unsigned.c`
6
7  ## `timing.c`
8
9  ## `char.c`
10
11 ## `int-overflow.c`
12
13 ## `implicit-inversion.c`
14
15 ## `explicit-inversion.c`
16
17 ## `float-limits.c`
18
19 ## `sums.c`
20
21 ## `loop.c`
22
23 ## `compare.c`
```

```c
//
// Created by hfwei on 2023/11/15.
//

#include <stdio.h>

#define LEN 7

/**
 * @brief Sort numbs[left .. right] using merge sort.
 * @param nums
 * @param left
 * @param right
 */
void MergeSort(int nums[], int left, int right);

/**
 * @brief Merge nums[left .. mid] and nums[mid + 1 .. right]
 * @param nums
 * @param left
 * @param mid
 * @param right
 */
void Merge(int nums[], int left, int mid, int right);

/**
 * @brief Copy src[left .. right] to dest[left .. right]
 * @param src
 * @param dest
 * @param left
 * @param right
 */
void Copy(const int src[], int dest[], int left, int right);

int main() {
  int numbers[LEN] = {38, 27, 43, 3, 9, 82, 10};
  MergeSort(numbers, 0, LEN - 1);

  for (int i = 0; i < LEN; i++) {
    printf("%d ", numbers[i]);
  }

  return 0;
}

void MergeSort(int nums[], int left, int right) {
  if (left == right) {
    return;
  }

  int mid = (left + right) / 2;
  MergeSort(nums, left, mid);      // Call the Mirror
  MergeSort(nums, mid + 1, right);  // Call the Mirror
```

```c

  Merge(nums, left, mid, right);
}

void Merge(int nums[], int left, int mid, int right) {
  static int copy[LEN] = {0};

  int left_index = left;
  int right_index = mid + 1;
  int copy_index = left;

  while (left_index <= mid && right_index <= right) {
    if (nums[left_index] <= nums[right_index]) {
      copy[copy_index] = nums[left_index];
      left_index++;
    } else {
      copy[copy_index] = nums[right_index];
      right_index++;
    }

    copy_index++;
  }

  while (left_index <= mid) {
    copy[copy_index] = nums[left_index];
    left_index++;
    copy_index++;
  }

  while (right_index <= right) {
    copy[copy_index] = nums[right_index];
    right_index++;
    copy_index++;
  }

  Copy(copy, nums, left, right);
}

void Copy(const int src[], int dest[], int left, int right) {
  for (int i = left; i <= right; ++i) {
    dest[i] = src[i];
  }
}
```

```c
// file: limits.h

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

int main() {
  printf("INT_MIN = %d\n", INT_MIN);
  printf("INT_MAX = %d\n\n", INT_MAX);

  printf("UINT_MIN = %u\n", 0U);
  printf("UINT_MAX = %u\n\n", UINT_MAX);

  printf("LONG_MIN = %ld\n", LONG_MIN);
  printf("LONG_MAX = %ld\n\n", LONG_MAX);

  printf("ULONG_MIN = %lu\n", 0UL);
  printf("ULONG_MAX = %lu\n\n", ULONG_MAX);

  // long long int: >= 64 bits
  printf("LLONG_MIN = %lld\n", LLONG_MIN);
  printf("LLONG_MAX = %lld\n\n", LLONG_MAX);

  printf("ULONG_LONG_MIN = %llu\n", 0ULL);
  printf("ULONG_LONG_MAX = %llu\n\n", ULONG_LONG_MAX);

  return 0;
}
```

```c
//
// Created by hfwei on 2022/11/10.
//

#include <stdio.h>

int main() {
  int array[] = {0, 1, 2, 3, 4};
  int i = -1;

  size_t size = sizeof array;
  printf("The size of the array is %zu\n", size);

  if (i <= size) {
    printf("i <= sizeof array\n");
  } else {
    printf("i > sizeof array\n");
  }

  return 0;
}
```

```c
//
// Created by hfwei on 2022/11/10.
//

#include <stdio.h>
#include <time.h>

long long Fib(int n);

int main() {
  int n;
  scanf("%d", &n);

  printf("Fib(%d) = %lld\n", n, Fib(n));

  return 0;
}

long long Fib(int n) {
  if (n <= 1) {
    return n;
  }

  return Fib(n - 1) + Fib(n - 2);
}
```

```c
//
// Created by hfwei on 2022/11/10.
//

#include <stdio.h>

int main() {
  // (signed) char on my computer: -128 ~ 127
  // using unsigned char c = 150;
  unsigned char c = 150;
  int i = 900;

  printf("i / c = %d\n", i / c);

  return 0;
}
```

```c
//
// Created by hfwei on 2022/11/10.
//

#include <stdio.h>
#include <limits.h>
int main() {
  printf("UINT_MAX = %u\n", UINT_MAX);

  unsigned int max = UINT_MAX;
  unsigned int one = 1U;
  unsigned int two = 2U;

  printf("max + one = %u\n", max + one);
  printf("one - two = %u\n", one - two);

  return 0;
}
```

```c
//
// Created by hfwei on 2022/11/10.
//

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

int SquareInt(int num);
double SquareDouble(double num);

int main() {
  // narrowing conversion (still in the range)

  // out of the range: undefined behavior!!!

  // arguments; narrowing conversion

  // return value; narrowing conversion

  return 0;
}

int SquareInt(int num) {
  return num * num;
}

double SquareDouble(double num) {
  return num * num;
}
```

```c
//
// Created by hfwei on 2022/11/10.
//

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

int main() {
  double pi = 3.14159;

  // below: obtain its fractional part
  double fraction = pi - (int) pi;

  int num = 1000000000;
  printf("LLONG_MAX = %lld\n", LLONG_MAX);
  // long long llint = num * num;
  // long long llint = (long long) num * num;
  long long llint = (long long) (num * num);

  printf("llint = %lld\n", llint);

  return 0;
}
```

```c
//
// Created by hfwei on 2022/11/9.
//

#include <stdio.h>
#include <float.h>

int main() {
  // 3.402823e+38
  printf("FLT_MAX = %e\n", FLT_MAX);
  // 1.175494e-38
  printf("FLT_MIN = %e\n", FLT_MIN);
  // 1.401298e-45
  printf("FLT_TRUE_MIN = %e\n", FLT_TRUE_MIN);
  // 1.192093e-07
  printf("FLT_EPSILON = %e\n\n", FLT_EPSILON);

  // %lf for scanf
  // 1.797693e+308
  printf("DBL_MAX = %e\n", DBL_MAX);
  // 2.225074e-308
  printf("DBL_MIN = %e\n", DBL_MIN);
  // 4.940656e-324
  printf("DBL_TRUE_MIN = %e\n", DBL_TRUE_MIN);
  // 2.220446e-16
  printf("DBL_EPSILON = %e\n\n", DBL_EPSILON);

  return 0;
}
```

```c
/**
 * file: sums.c
 * See https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/
 *
 * Created by hengxin on 11/21/21.
 */

#include <stdio.h>

int main() {
  // 0.1: 0.0 0011 0011 0011
  float f = 0.1F;
  float sum = 0.0F;

  for (int i = 0; i < 10; ++i) {
    sum += f;
  }

  float product = f * 10;

  printf("sum = %.15f\nmul = %.30f\n",
         sum, product);

  return 0;
}
```

```c
/**
 * file: loop.c
 *
 * Created by hengxin on 11/21/21.
 */

#include <stdio.h>

int main() {
  /**
   * Do not use a counter of type float/double,
   * although it works on some platforms.
   *
   * 0.1 cannot be exactly represented in machines.
   */
  for (double x = 0.1; x <= 1.0; x += 0.1) {
    printf("%.20f\n", x);
  }

  return 0;
}
```

```c
/**
 * file: compare.c
 *
 * See https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/
 *
 * Created by hfwei on 2022/11/10.
 */

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

bool IsEqual(double x, double y);

int main() {
  printf("%d\n", IsEqual(DBL_MAX, DBL_MAX - 100));

  printf("%.50f\n", DBL_MAX - (DBL_MAX - 100));

  return 0;
}

bool IsEqual(double x, double y) {
  return fabs(x - y) <= DBL_EPSILON;
}
```

```c
//
// Created by hfwei on 2022/11/10.
//

#include <stdio.h>
#include <time.h>

long long Fib(int n);

int main() {
  int n;
  scanf("%d", &n);

  time_t start = time(NULL);

  printf("Fib(%d) = %lld\n", n, Fib(n));

  time_t end = time(NULL);
  printf("Time elapsed: %lld seconds\n", (long long) end - start);

  return 0;
}

long long Fib(int n) {
  if (n <= 1) {
    return n;
  }

  return Fib(n - 1) + Fib(n - 2);
}
```

```c
//
// Created by hfwei on 2022/11/10.
//

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

int SquareInt(int num);
double SquareDouble(double num);

int main() {
  // narrowing conversion (still in the range)
  int i = 3.14159;

  // out of the range: undefined behavior!!!
  int j = UINT_MAX;

  // arguments; narrowing conversion
  double k = 3.14159;
  SquareInt(k);

  // return value; narrowing conversion
  int m = SquareDouble(k);

  int big = 1234567890;
  float approx = big;
  int approx_big = (int) approx;
  printf("big = %d\t approx = %f\t approx_big = %d\t diff = %d\n",
         big, approx, approx_big, big - (int) approx);

  return 0;
}

int SquareInt(int num) {
  return num * num;
}

double SquareDouble(double num) {
  return num * num;
}
```