智能工程学院 C 语言程序设计考前拾遗

编译过程

$$\text{main.c} \xrightarrow{\text{编译}} \text{main.obj} \xrightarrow{\text{链接}} \text{main.exe}$$

数据类型

数据类型	占用字节数	可表示数字范围（有符号）	可表示数字范围（无符号）
`char`	1	−128 到 127	0 到 255
`unsigned char`	1	—	0 到 255
`short`	2	−32,768 到 32,767	0 到 65,535
`unsigned short`	2	—	0 到 65,535
`int`	4	−2,147,483,648 到 2,147,483,647	0 到 4,294,967,295
`unsigned int`	4	—	0 到 4,294,967,295
`long`	4 或 8	−2,147,483,648 到 2,147,483,647（32 位） 或 −9,223,372,036,854,775,808 到 9,223,372,036,854,775,807（64 位）	0 到 4,294,967,295（32 位） 或 0 到 18,446,744,073,709,551,615（64 位）
`unsigned long`	4 或 8	—	0 到 4,294,967,295（32 位） 或 0 到 18,446,744,073,709,551,615（64 位）
`long long`	8	−9,223,372,036,854,775,808 到 9,223,372,036,854,775,807	0 到 18,446,744,073,709,551,615
`unsigned long long`	8	—	0 到 18,446,744,073,709,551,615
`float`	4	约 ±3.4e-38 到 ±3.4e38	—
`double`	8	约 ±1.7e-308 到 ±1.7e308	—
`long double`	8 或 16	范围更大，具体取决于实现	—

输入输出

格式化：`printf()` `scanf()`

主要是 `printf()` 函数的格式化输出。

占位符	说明	示例
`%d`	输出有符号十进制整数	`printf("%d", 42);`
`%u`	输出无符号十进制整数	`printf("%u", 42);`
`%o`	输出无符号八进制整数	`printf("%o", 42);`
`%x`	输出无符号十六进制整数（小写）	`printf("%x", 42);`
`%X`	输出无符号十六进制整数（大写）	`printf("%X", 42);`
`%f`	输出浮点数（默认 6 位小数）	`printf("%f", 3.14);`
`%e`	输出科学计数法（小写）	`printf("%e", 3.14);`
`%E`	输出科学计数法（大写）	`printf("%E", 3.14);`
`%g`	自动选择 `%f` 或 `%e`（更简洁）	`printf("%g", 3.14);`
`%c`	输出单个字符	`printf("%c", 'A');`
`%s`	输出字符串	`printf("%s", "Hello");`
`%p`	输出指针地址	`printf("%p", &var);`
`%%`	输出百分号 `%`	`printf("%%");`

修饰符	说明	示例
%Nd	最小宽度为 N，不足时用空格填充	printf("%5d", 42);
%0Nd	最小宽度为 N，不足时用 0 填充	printf("%05d", 42);
%.Mf	浮点数保留 M 位小数	printf("%.2f", 3.14159);
%*.*f	动态指定宽度和精度（通过参数传递）	printf("%*.*f", 5, 2, 3.14);
%-Nd	左对齐（默认右对齐）	printf("%-5d", 42);
%+d	显示正负号	printf("%+d", 42);
%#x	显示十六进制前缀 0x	printf("%#x", 42);

错误处理：

- scanf() 成功时返回成功解析并赋值的输入项的数量（≥ 1）
- printf() 成功时返回成功输出的字符总数（包括空格、换行符等）（≥ 1）

其他输入输出

- gets() 读入一行，含换行符
- getchar() 从缓冲区获取一个字符
- puts() 输出一行
- putchar() 输出一个字符

运算符

- 逗号运算符：(value1, value2) 的返回值是 value2
- 逻辑运算符：! > && > ||
- 关系运算符：不等号 > 等号 > 赋值

变量的生存周期

- 用户区：

1. 程序区
2. 静态储存区
3. 动态储存区
   - 堆：手动管理的动态内存如 int *a = malloc(5*4)
   - 栈：编译器管理的内存，如 int a[5]

- extern：外部变量
- static：静态变量
  1. **只会被初始化一次**
  2. 相当于 internal

一些常用的库函数

字符串：string.h

- size_t strlen(const char *str) 统计不含 \0 的长度
- int strcmp(const char *str1, const char *str2)
  - 把 str1 所指向的字符串和 str2 所指向的字符串进行比较
    - 如果返回值小于 0，则表示 str1 小于 str2
    - 如果返回值大于 0，则表示 str1 大于 str2
    - 如果返回值等于 0，则表示 str1 等于 str2
  - 即只有返回值为 0 的时候，str1 和 str2 相等
- char *strtok(char *str, const char *delim) 分割字符串，返回第一个子串

```c
#include <string.h>
#include <stdio.h>

int main () {
```

```
   char str[80] = "This is - www.runoob.com - website";
   const char s[2] = "-";
   char *token;

   /* 获取第一个子字符串 */
   token = strtok(str, s);

   /* 继续获取其他的子字符串 */
   while( token != NULL ) {
      printf( "%s\n", token );
      token = strtok(NULL, s);
   }
   return 0;
}
```

- `char *strcpy(char *dest, const char *src)` 复制字符串

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

int main()
{
   char src[40];
   char dest[100];

   memset(dest, '\0', sizeof(dest));
   strcpy(src, "This is runoob.com");
   strcpy(dest, src);

   printf("最终的目标字符串： %s\n", dest);

   return(0);
}
```

数学：`math.h`

基本都返回 `double` 类型：`fabs()`, `floor()`, `ceil()`, `sqrt()`, `pow()`, `log()`, `log10()`

- `double modf(double x, double *integer)`

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

int main ()
{
   double x, fractpart, intpart;

   x = 8.123456;
   fractpart = modf(x, &intpart);

   printf("整数部分 = %lf\n", intpart);
   printf("小数部分 = %lf \n", fractpart);

   return(0);
}
```

有用的杂项：`<stdlib.h>`

- `Type atoT(const char *str)` 将字符串转换为对应类型
  - T: f → `double`, i → `int`, f → `double`, etc.
- `qsort()`

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

int comp(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);
}
```

```c
int main() {
    int arr[] = {5, 2, 3, 1, 4};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Sort the array arr
    qsort(arr, n, sizeof(arr[0]), comp);

    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

内存管理:

- ```c
  void* malloc(size_t size);
  void* memset(void* ptr, int value, size_t num);
  void free(void* ptr);
  ```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    int *arr;
    int n = 5;

    // 分配内存
    arr = (int*)malloc(n * sizeof(int));
    if (arr == NULL) {
        printf("内存分配失败\n");
        return 1;
    }
    // 使用 memset 初始化内存
    memset(arr, 0, n * sizeof(int));
    // 打印数组
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
    // 释放内存
    free(arr);

    return 0;
}
```

一些简单的算法

递归

```c
#include <stdio.h>

// 递归函数，解决汉诺塔问题
void hanoi(int n, char from, char to, char aux) {
    if (n > 0) {
        hanoi(n - 1, from, aux, to); // 将 n-1 个盘子从 from 移动到 aux
        printf("Move disk %d from %c to %c\n", n, from, to); // 将第 n 个盘子从 from 移动到 to
        hanoi(n - 1, aux, to, from); // 将 n-1 个盘子从 aux 移动到 to
    }
}

int main() {
    int n = 3; // 盘子的数量
    hanoi(n, 'A', 'C', 'B'); // A 是起始柱，C 是目标柱，B 是辅助柱
    return 0;
}
```
```

冒泡排序 Bubble

```c
#include <stdio.h>
int main()
{
    int a[]=  {3, 5, 1, 88 , 2, 45, 23, 67, 89, 12};
    for (int i = 0; i < 10; i++)
    {
        for (int j = 0; j < 10 - i - 1; j++)
        {
            if (a[j] > a[j + 1])
            {
                int t = a[j];
                a[j] = a[j + 1];
                a[j + 1] = t;
            }
        }
    }
    for (int i = 0; i < 10; i++)
    {
        printf("%d ", a[i]);
    }
    return 0;
}
```

选择排序

```c
#include <stdio.h>
int main()
{
    int a[]=  {3, 5, 1, 88 , 2, 45, 23, 67, 89, 12};
    int min = 0;
    for (int i = 0; i < 10; i++)
    {
        for(int j = i; j < 10; j++)
        {
            if(a[j] < a[min])
            {
                min = j;
            }
        }
        int temp = a[i];
        a[i] = a[min];
        a[min] = temp;
        min = i + 1;
    }
    for (int i = 0; i < 10; i++)
    {
        printf("%d ", a[i]);
    }
    return 0;
}
```