Lab 4

Task 1: 指针基本特性

- 类型是分配内存块大小的别名，即类型（int,double,char）的作用就是分配相对应大小的内存并给程序员一个名字（int,double,char）方便操作。
- 指针也是一种数据类型，定义时可以对其赋值（可赋任意地址值，但习惯赋值为NULL，方便操作管理）。
- C语言允许对NULL地址操作，而不会产生错误或者任何效果。
- 在C语言中, sizeof() 是一个判断数据类型或者表达式长度的运算符。sizeof并不是函数，而是一种编译指令，对sizeof的求值发生在编译器。

编写程序，查看 int, double, char 型指针的长度。（`task1.c`）

Task 2: 指针数组

本示例代码见 `task2.c` 。

`char *a[]={"Hello","GNUC","world"};`

- `char *a[]`：表示a是数组，数组中的元素是指针，指向char类型。数组里面所有的元素是连续的内存存放的。
- 数组名是数组第一个字节的内存地址，并且数组名a也表示指针。a并不表示a地址存储的内容, 而是a地址本身。
- 我们注意到，因为a的元素是char指针, 所需要的空间为8字节(64位内存地址)，那么：
  - `a+1`：表示a的第二个元素（char 指针）的内存地址, 所以是加 8 字节；
  - `*(a+1)`：则表示a这个数组的第二个元素的内容 (是个char 类型的指针，本例表示为GNUC字符串的地址)；
  - `*(*(a+1))`：则表示a这个数组的第二个元素的内容(char指针)所指向的内容(G字符)；
  - `char * a[3]`：表示限定这个数组最多可存放3个元素(char指针), 也就是说这个数组占用3*8 = 24字节;

思考：`a[0]+1` 和 `a+1`是同一件事吗？请做做实验试试看。

```
printf("%c\n",*(a[0]+1));    //! e
printf("%s\n",*(a+1));       //! GNUC
```

Task 2.1

分析下面代码输出情况。

```
printf("a[0]:       %p\n",(a[0]));      //!
printf("a[0]+1:     %p\n",(a[0]+1));    //!
printf("a[1]:       %p\n",(a[1]));      //!
printf("(a+1)[0]:   %p\n",(a+1)[0]);    //!
printf("a:          %p\n",(a));         //!
printf("a+1:        %p\n",(a+1));       //!
```

Task 2.2

填写对应的缺失值，使得输出指定字符。

```c
    printf("%s\n",      );      //! Hello
    printf("%s\n",      );      //! Hello

    printf("%s\n",      );      //! ello
    printf("%s\n",      );      //! ello

    // remind:
    printf("%s\n",      );      //! GNUC
    printf("%s\n",      );      //! GNUC
    printf("%c\n",      );      //! G

    printf("%c\n",      );      //! d
    printf("%c\n",      );      //! d
```

Task 3: 洗牌发牌例子

下面是一个洗牌发牌的程序案例（task3.c）：

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

void shuffle(int[][13]);
void deal(const int[][13], const char *[], const char *[]);

int main()
{
    const char *suit[4] = {"Hearts", "Diamonds", "Clubs", "Spades"};
    const char *face[13] = {"Ace", "Deuce", "Three", "Four",
                            "Five", "Six", "Seven", "Eight",
                            "Nine", "Ten", "Jack", "Queen", "King"};
    int deck[4][13] = {0};

    srand(time(0));

    shuffle(deck);
    deal(deck, face, suit);

    return 0;
}

void shuffle(int wDeck[][13])   // Q1-4
{
    int row, column, card;

    for (card = 1; card <= 52; card++)
    {
        do
        {
            row = rand() % 4;
            column = rand() % 13;
        } while (wDeck[row][column] != 0);

        wDeck[row][column] = card;
    }
}

void deal(const int wDeck[][13], const char *wFace[], const char *wSuit[])  // Q5-6
{
    int card, row, column;

    for (card = 1; card <= 52; card++)
        for (row = 0; row <= 3; row++)
            for (column = 0; column <= 12; column++)
                if (wDeck[row][column] == card)
                    printf("%5s of %-8s%c",
                           wFace[column], wSuit[row],
                           card % 2 == 0 ? '\n' : '\t');
}
```

Q1: `shuffle` 函数的参数 `int wDeck[][13]` 是什么意思?

Q2: `int wDeck[][13]` 还可以写成什么形式?

Q3: 为什么不直接使用 `int wDeck[4][13]` 作为参数?

Q4: `shuffle` 函数洗牌的原理是什么?

Q5: `deal` 函数的参数 `const int wDeck[][13], const char *wFace[], const char *wSuit[]` 中出现了三个

const。这三个const的作用是什么?

Q6: 如果希望指针本身指向的地址保持不变，但是指针指向的值可以修改，应该如何声明?

Task 4: 函数指针

函数指针：函数在编译时被分配的入口地址,用函数名表示。

函数指针指向的是程序代码存储区。

用函数指针变量调用函数。

函数指针变量定义形式：

1. 数据类型 (*指针变量名)( ); 如 int (*p)( )
2. 函数指针变量赋值：如 p=max
3. 函数调用形式： c=max(a,b); c=(*p)(a,b);
4. 注意：对函数指针变量 p±n，p++，p-- 无意义

求a和b中的最大者，使用以下几种方法：

Task 4.1

一般方法。（task4_1.c）

```c
#include <stdio.h>

int max(int, int);

int main()
{
    int a, b, c;
    scanf("%d %d", &a, &b);
    c = max(a, b);
    printf("a=%d,b=%d,max=%d\n", a, b, c);
    return 0;
}

int max(int x, int y)
{
    int max_number;
    // TODO

    return max_number;
}
```

Task 4.2

通过指针变量访问函数（task4_2.c）

```c
#include <stdio.h>

int max(int, int);
int (*p)(int, int);

int main()
{
    int a, b, c;
    scanf("%d %d", &a, &b);
    // TODO

    printf("a=%d,b=%d,max=%d\n", a, b, c);
    return 0;
}

int max(int x, int y)
{
    int max_number;
    if (x > y)
        max_number = x;
    else
        max_number = y;
    return max_number;
}
```

Task 4.3

用函数指针变量作参数，求最大值、最小值和两数之和。（task4_3.c）

函数指针变量通常用途是将指针作为参数传递到其他函数，实现对不同函数的调用。

```c
#include <stdio.h>

int max(int, int);
int min(int, int);
int add(int, int);
int process(int, int, int (*fun)(int, int));

int main()
{
    int a, b;
    printf("enter a and b:");
    scanf("%d %d", &a, &b);
    // TODO

    return 0;
}

int process(int x, int y, int (*fun)(int, int))
{
    int result;
    result = (*fun)(x, y);
    printf("%d\n", result);
    return 0;
}

int max(int x, int y)
{
    printf("max=");
    return (x > y ? x : y);
}

int min(int x, int y)
{
    printf("min=");
    return (x < y ? x : y);
}

int add(int x, int y)
{
    printf("sum=");
    return (x + y);
}
```

Task 5: Homework

回忆一下上节课回文字符串判断函数，首次尝试了对字符串的操作。

```c
//! 判断是否是 Palindrome array
int isPalindromic(char *arr, int len)
{
    for (int i = 0; i < len / 2; i++)
    {
        if (arr[i] != arr[len - i - 1])
        {
            return 0;
        }
    }
    return 1;
}
```

本节课，我们尝试用函数指针、字符串指针的知识，复现C语言中的strcpy()函数。

strcpy()函数：是将一个字符串复制到另一块空间地址中的函数，`'\0'`是停止拷贝的终止条件，同时也会将 `'\0'` 也复制到目标空间。

The `strcpy` function copies **strSource**, including the terminating null character, to the location specified by **strDestination**. No overflow checking is performed when strings are copied or appended. The behavior of `strcpy` is undefined if the source and destination strings overlap.

Task 5.1

编程实现 `char *strcpy( char *strDestination, const char *strSource)` 。

- `char *strDestination` : 目标地址；
- `const char *strSource` : 源地址；
- 返回值：操作成功，返回目标地址，否则返回NULL；

尝试用自己编写的strcpy函数或内置函数试验下面的代码：

(task5.c)

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

int main(void)
{
    char buff[8] = {0};
    char *p = "0123456789";
    strcpy(buff, p);
    printf("%s\n", buff);
    return 0;
}
```

检验是否发生缓冲区溢出，并分析原因。

- **缓冲区**：是指程序运行期间，在内存中分配的一个连续的区域，用于保存包括字符数组在内的各种数据类型；
- **溢出**：所填充的数据超出了原有的缓冲区边界，并非法占据了另一段内存区域；
- **缓冲区溢出与黑客攻击**：由于填充数据越界而导致原有流程的改变，黑客借此精心构造填充数据，让程序转而执行特殊的代码，最终获取控制权；

Task 5.2

根据5.1的分析，升级 `strcpy` 函数到 `strncpy` ，修复缓冲区溢出bug。

作业要求

1. 提交代码
    - 补全 Task 4：TODO部分的代码
    - 完成 Task 5：不需要scanf读取输入，在main函数中打印：
        1. 实现的 `strcpy` 的正常使用的case；
        2. 缓冲区溢出的case；
        3. `strncpy` 解决缓冲区溢出case后的结果；
2. 提交报告
    - 解释Task 5核心代码部分，分析bug原因以及解决方案。

参考资料

- strcpy：https://blog.csdn.net/m0_62179366/article/details/123098864
- 缓冲区溢出：https://blog.csdn.net/hyb612/article/details/83868330
- Strncpy：https://blog.csdn.net/m0_65601072/article/details/123974984
- 缓冲区溢出攻击：
    - https://blog.csdn.net/VN520/article/details/130283059
    - https://baijiahao.baidu.com/s?id=1692690710869565236&wfr=spider&for=pc