**数据结构**

# 1、抽象数据类型（ADT: abstract data type）

可以用数组实现动态列表，但对于内存的利用率和消耗来说它不是最有效的（根据时间复杂度得出的结论）。因为在申明数组是就固定了数组的大小，如果后续需要更大的数组，就需要重新创建数组，并且把钱一个数组的内容全部复制过去，然后再把之前的那一块内存释放。因此需要一种更加有效的方式---链表

# 2、linked-list

内存块的地址是第一个字节的地址，类似的函数指令的地址是第一句指令的地址。对于数组来说是一个连续的内存块存储在内存中。

#include <stdio.h>

int main(int arge,const char\* argv)

{

    int a = 5;

    printf("%p\n",&a);

    printf("%p\n",&a+1);

    char\* p = (char\*)&a;

    printf("%p\n",p+1);

    printf("hello world!\n");

    printf("%d\n",sizeof(a));

    return 0;

}

对于这段代码就可以看出来，一个整型的地址是第一个字节的地址。并且对于&a+1代表&a+1\*sizeof(type(a))。

链表：

struct node

{

    int data;

    struct node\* next;

};

以上整体称为一个节点。next为一个指针，指向下一节点。链表有一个头指针head指向第一个节点，最后的一个指针指向NULL。

以下代码用于验证，一个结构体所占字节，以及结构体内部是怎样实现内存分配，字节填充。

#include <stdio.h>

typedef struct \_\_attribute\_\_((packed)) node

{

    int data;

    struct node\* next;

}node1;

node1 a;

int main(int arge,const char\* argv)

{

    // int a = 5;

    printf("%d\n",sizeof(node1));

    printf("%d\n",sizeof(a.data));

    printf("%d\n",sizeof(a.next));

    printf("%d\n",sizeof(a));

    printf("%d\n",&a);

    printf("%d\n",&a+1);

    char\* p = (char\*)&a;

    printf("%p\n",p+1);

    printf("hello world!\n");

    printf("%d\n",sizeof(a));

    return 0;

}

\_\_attribute\_\_((packed))，这个函数用于设置禁止字节填充，在某些情况下，编译器会为了对齐数据而在结构体成员之间插入一些填充字节，以保证结构体的每个成员都按照某种对齐规则放置。

# 3、linked-list compared with array

对于数组和链表各有各的优势：

1）对于直接访问某个元素，数组显然要更加快捷，时间复杂度为O(1)，而对于linked-list来说时间复杂度为O(n)。

2）对于插入/删除某个特定的元素来说，时间复杂度数组和链表在不同地方时间复杂度不一样。

3）对于内存利用率来说，内存的使用情况链表更高，但也对应有额外数值用于存储下个数据的地址。但链表可变长短，操作性更高。

4）一个数据加一个指向下一个数据的指针称为节点，而只向下一个数据的指针称为链接点。

1.4、linked-list implementation in C

#include <stdio.h>

#include <stdlib.h>

typedef struct \_\_attribute\_\_((packed)) node

{

    int data;

    struct node\* next;

}node1;

int main(int arge,const char\*\* argv)

{

    node1\* a = (node1\*)calloc(sizeof(node1),1);

    a->data = 2;

    a->next = (node1\*)calloc(sizeof(node1),1);

    a->next->data = 4;

    a->next->next = (node1\*)calloc(sizeof(node1),1);

    a->next->next->data = 6;

    a->next->next->next = NULL;

    printf("%d\n",a->data);

    printf("%d\n",(\*a).data);

    printf("%d\n",a->next);

    printf("%d\n",a->next->data);

    printf("%d\n",a->next->next);

    printf("%d\n",a->next->next->data);

    printf("%d\n",a->next->next->next);

    printf("%d\n",sizeof(node1));

    free(a->next->next);

    free(a->next);

    free(a);

}

通过不断的指向下一个结构体，实现数据的存储。对于printf(“%d\n”,a->data)这里a->data是指一个数值，表示通过指针a访问结构体成员data的值，这个值的类型是int，而不是指针，等价于(\*a).data。记得释放内存，和多维数组一样，从里到外依次释放。

用while进行遍历

#include <stdio.h>

#include <stdlib.h>

typedef struct \_\_attribute\_\_((packed)) node

{

    int data;

    struct node\* next;

}node1;

int main(int arge,const char\*\* argv)

{

    node1\* a = (node1\*)calloc(sizeof(node1),1);

    a->data = 2;

    a->next = (node1\*)calloc(sizeof(node1),1);

    a->next->data = 4;

    a->next->next = (node1\*)calloc(sizeof(node1),1);

    a->next->next->data = 6;

    a->next->next->next = (node1\*)calloc(sizeof(node1),1);

    a->next->next->next->data = 0;

    a->next->next->next->next = NULL;

    node1\* temp = a;

    do

    {

        printf("%d\n",temp->data);

        // printf("%d\n",(\*temp).data);

        printf("%d\n",temp->next);

        temp = temp->next;

    }while(temp->next);

    printf("%d\n",sizeof(node1));

    free(a->next->next);

    free(a->next);

    free(a);

}

注意不要移动a指针，他是一个头指针，不能丢，否则就找不到他移动后之前的信息了。头指针永远不能变。

# 4、在链表的开头插入数据：

#include <stdio.h>

#include <stdlib.h>

#include "main.h"

typedef struct node

{

    int data;

    struct node\* next;

}node1;

node1\* head = NULL;

void insert(int x);

void print();

int main(int arge,const char\*\* argv)

{

    printf("please input number about your wanner create linked-list\n");

    int n,x;

    scanf("%d",&n);

    for(int i = 0;i<n;i++)

    {

        printf("enter the number\n");

        scanf("%d",&x);

        insert(x);

        print();

    }

    return 0;

}

void insert(int x)

{

    node1\* temp = (node1\*)malloc(sizeof(node1));

    temp->data = x;

    temp->next = head;

    head = temp;

}

void print()

{

    node1\* temp = head;

    printf("the list is: ");

    while(temp)

    {

        printf("%d ",temp->data);

        // printf("%d\n",temp->next);

        temp = temp->next;

    }

    printf("\n");

}

注意其中对于链表开始插入数据的逻辑关系。防止数据丢失。此时的head为全局变量。

修改，此时head为局部变量，但每个里面都有一个局部变量head。并且注意此时的insert函数的返回值为指针。

#include <stdio.h>

#include <stdlib.h>

#include "main.h"

typedef struct node

{

    int data;

    struct node\* next;

}node1;

node1\* insert(node1\* head,int x);

void print(node1\* head);

int main(int arge,const char\*\* argv)

{

    node1\* head = NULL;

    printf("please input number about your wanner create linked-list\n");

    int n,x;

    scanf("%d",&n);

    for(int i = 0;i<n;i++)

    {

        printf("enter the number\n");

        scanf("%d",&x);

        head = insert(head,x);

        print(head);

    }

    return 0;

}

node1\* insert(node1\* head,int x)

{

    node1\* temp = (node1\*)malloc(sizeof(node1));

    temp->data = x;

    temp->next = head;

    head = temp;

    return head;

}

void print(node1\* head)

{

    printf("the list is: ");

    while(head)

    {

        printf("%d ",head->data);

        // printf("%d\n",temp->next);

        head = head->next;

    }

    printf("\n");

}

对上面算法进一步修改，使得head为main函数中的局部变量，其他都是在这个地址上进行操作。

#include <stdio.h>

#include <stdlib.h>

#include "main.h"

typedef struct node

{

    int data;

    struct node\* next;

}node1;

void insert(node1\*\* head,int x);

void print(node1\* head);

void freememory(node1\* head);

int main(int arge,const char\*\* argv)

{

    node1\* head = NULL;

    printf("please input number about your wanner create linked-list\n");

    int n,x;

    scanf("%d",&n);

    for(int i = 0;i<n;i++)

    {

        printf("enter the number\n");

        scanf("%d",&x);

        insert(&head,x);

        print(head);

    }

    freememory(head);

    return 0;

}

void insert(node1\*\* head,int x)

{

    node1\* temp = (node1\*)malloc(sizeof(node1));

    temp->data = x;

    temp->next = \*head;

    \*head = temp;

}

void print(node1\* head)

{

    printf("the list is: ");

    while(head)

    {

        printf("%d ",head->data);

        // printf("%d\n",temp->next);

        head = head->next;

    }

    printf("\n");

}

void freememory(node1\* head)

{

    node1\* current = head;

    node1\* next;

    while(current)

    {

        next = current->next;

        free(current->next);

        current = next;

    }

}

此时head为main函数中的局部变量，在insert函数中时，传递了指向main函数局部变量head的地址，来进行处理。而对于print函数来说，不需要用head的地址来进一步处理了。当然也可以那么做，但直接传递head给print函数更加省事。最后一段为内存释放语句，采用逐级释放。