

18 高階指標應用

18.1 指標與字串

在字串這一章中，我們已經介紹過兩種C語言的字串：字串陣列與字串指標，請參考下面的程式：

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

int main()
{
    char str[]="Hello";
    char *p;
    int i;

    printf("str at %p\n", str);

    for(i=;i<strlen(str);i++)
    {
        printf("str[%d] at %p\n", i, &str[i]);
    }

    p = str;

    for(i=;i<strlen(str);i++)
    {
        printf("(p+%d)=str[%d]=%c at %p\n", i, i, *(p+i), p+i);
    }

    str[]='h';
    *(p+3)='L';

    puts(str);
    puts(p);

    return ;
}
```

在這個程式中，我們先宣告了一個字串陣列，然後讓指標p指向該字串，我們可以使用陣列與指標的方式來存取在記憶體中的這個字串。其實這兩種方式在記憶體中是使用同樣的配置，其差別只在於我們是以陣列的索引來存取，亦或是使用指標來存取。換句話說，一個字串陣列可以當成指標字串來使用，反之亦然。請參考下面的程式：

[string2.c](#)

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

int main()
{
    char str[]="Hello";
    char *p;

    p = str;

    p[]='H';
    *(str+3) = 'L';

    puts(str);
    puts(p);

    return ;
}
```

18.2 動態配置字串

我們也可以使用「`malloc()`函式」或「`calloc()`函式」來動態地在記憶體中配置字串所需的空間，例如：

```
#define LEN 10;

char *str = malloc((LEN+1)*sizeof(char));
或是
char *str = malloc(LEN+1);
```

但是要特別注意的是，這種動態配置的字串，從其配置開始至程式結束，都會一直存在記憶體中，除非我們以「`free()`函式」將之釋放。為了在程式執行時，不要過度佔用記憶體，當我們不再需要該字串時，應該以下列程式碼將其釋放：

```
free(str);
```

如果要動態地宣告由多個字串所組成的陣列，又該如何做呢？請參考下面的例子：

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

#define numString 10
#define LEN 20

int main()
{
    char *strs[numString];
    int i;
```

```
for(i=;i<numString;i++)
{
    strs[i] = (char *) (malloc(LEN+1));
}

for(i=;i<numString;i++)
{
    printf("String %d = ", i+1);
    scanf(" %[^\\n]", strs[i]);
}

for(i=;i<numString;i++)
{
    puts(strs[i]);
}

return ;
}
```

我們也可以將上述程式，改成以指向指標的指標的方式來完成：

stringArray2.c

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

#define numString 10
#define LEN 20

int main()
{
    char **strs;
    int i;

    strs = malloc(numString*sizeof(void *));
    for(i=;i<numString;i++)
        *(strs+i) = malloc(LEN+1);

    for(i=;i<numString;i++)
    {
        printf("String %d = ", i+1);
        scanf(" %[^\\n]", *(strs+i));
    }

    for(i=;i<numString;i++)
        puts(*(strs+i));

    return ;
}
```

```
}
```

18.3 字串操作函式之實作

我們以下列範例示範以指標進行各式的字串操作：

計算字串長度

首先是實作計算字串長度的函式：

```
int strlen(const char *s)
{
    int n;
    for(n=; *s!='\0';s++)
        n++;

    return n;
}
```

```
int strlen(const char *s)
{
    int n=;
    for(; *s!='\0';s++)
        n++;

    return n;
}
```

```
int strlen(const char *s)
{
    int n=;
    for(; *s; s++)
        n++;

    return n;
}
```

```
int strlen(const char *s)
{
    int n=;
    for(; *s++; )
        n++;

    return n;
}
```

```
int strlen(const char *s)
{
    int n=;

    while(*s++)
        n++;

    return n;
}
```

```
int strlen(const char *s)
{
    const char *p = s;

    while(*s)
        s++;

    return s-p;
}
```

字串串接

```
char *strcat(char *s1, const char *s2)
{
    char *p = s1;
    while(*p != '\0')
        p++;
    while(*s2 != '\0')
    {
        *p = *s2;
        p++;
        s2++;
    }
    *p = '\0';
    return s1;
}
```

```
char *strcat(char *s1, const char *s2)
{
    char *p = s1;

    while(*p)
        p++;
    while(*p++ = *s2++);
    return s1;
}
```

trim

設計一個trim()函式，將字串前與後的空白字元移除：

```
char *trim(char *s)
{
    char *f = s;
    char *t = s;

    while(*t != '\0')
    {
        if(*f == ' ')
            f++;
        t++;
    }

    t--;
    while((*t == ' ') || (*t == '\n'))
    {
        t--;
    }
    t++;
    *t = '\0';

    s = f;
    return s;
}
```

18.4 動態陣列

下面這個程式利用動態配置的方法，建立了一個陣列，並在執行時間改變其大小：

[dynarray.c](#)

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

int main()
{
    int *data;
    int size, i;

    printf("Array Size=? ");
    scanf(" %d", &size);

    data = malloc(size*sizeof(int));
```

```
for(i=i<size;i++)
    data[i]=i;

// add 10 more numbers
data = realloc(data, (size+10)*sizeof(int));
for(i=i<size+10;i++)
{
    data[i]=i;
}

// remove last 5 numbers
data = realloc(data, (size+5)*sizeof(int));

for(i=i<size+5;i++)
    printf("%d ", data[i]);
printf("\n");

return ;
}
```

下面的例子，則是動態建置一個二維的陣列：

[dynarray2d.c](#)

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

int main()
{
    int **data;
    int Row, Col;
    int i,j;

    Row=3;
    Col=2;

    data = malloc(sizeof(int *)*Row);
    for(i=i<Row;i++)
        data[i] = malloc(Col*sizeof(int));

    for(i=i<Row;i++)
        for(j=j<Col;j++)
            data[i][j]=i*Col+(j+1);

    for(i=i<Row;i++)
    {
        for(j=j<Col;j++)
            printf("%d ", data[i][j]);
        printf("\n");
    }
}
```

```
    return ;  
}
```

在這個例子中，`data`在語法上是一個指向整數的指標的指標(pointer to pointer)但在語意上是一個以指標來操作的陣列，其中每個陣列的元素為一個以指標來操作的陣列，也就形成了一個二維陣列。我們稱此種「雙指標」為「pointer to pointers」

18.5 動態結構體

下面這個程式動態產生了一個Point的結構體：

[dynStruct.c](#)

```
#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct  
{  
    int x;  
    int y;  
} Point;  
  
void showPoint(Point p)  
{  
    printf("(%d,%d)\n", p.x, p.y);  
}  
  
int main()  
{  
    Point *p1 = malloc(sizeof(Point));  
    p1->x=5;  
    p1->y=10;  
    showPoint(*p1);  
    return ;  
}
```

我們也可以產生一個動態的陣列用以存放多個結構體：

[dynStructArray.c](#)

```
#include <stdio.h>  
#include <stdlib.h>  
  
typedef struct  
{  
    int x;
```



```

    int y;
} Point;

void showPoint(Point p)
{
    printf("(%d,%d)\n", p.x, p.y);
}

#define LEN 10

int main()
{
    Point *ps = malloc(sizeof(Point)*LEN);
    int i;

    for(i=0;i<LEN;i++)
    {
        ps[i].x=i;
        ps[i].y=i;
        showPoint(ps[i]);
        // showPoint(*(ps+i));
    }
    return ;
}

```

18.6 函式指標

函式指標(function pointer)為指向函式的指標，在C語言裡，若使用函式名稱但不接後續的括號，就會被視為是一個指向該函式的指標(也就是該函式在記憶體中的位址)，請參考下列：

```

int foo(int f)
{
    return f*f;
}

int main()
{
    printf("Function foo at %p.\n", foo);
    printf("The address of Function foo is %p.\n", &foo);
    return ;
}

```

請執行上述的程式，看看其結果。就如同陣列`int data[]`其「`data`」與「`&data`」的值是一樣的，使用`foo`與「`&foo`」都是代表`foo`函式在記憶體中的位址，因此，我們可以宣告一個指標指向該位址：

```

int (*f)(int); //其中第一個int是函式的傳回值，第二個int則是引數

```

我們可以在程式中以下列程式碼，讓`f`指向`foo`函式」，並加以呼叫：

```
f=foo;  
  
printf("%d\n", f(3));  
或是  
printf("%d\n", (*f)(3));
```

這樣一來，在程式執行時，我們也可以讓指標動態地指向不同的函式，以完成不同的操作。請參考下面的程式：

funpointer.c

```
#include <stdio.h>  
#include <math.h>  
  
int maximum(int d[], int n)  
{  
    int max=d[0];  
    int i;  
  
    for(i=1;i<n;i++)  
        if(max<d[i])  
            max=d[i];  
    return max;  
}  
  
int minimum(int d[], int n)  
{  
    int min=d[0];  
    int i;  
  
    for(i=1;i<n;i++)  
        if(min>d[i])  
            min=d[i];  
    return min;  
}  
  
int median(int d[], int n)  
{  
    double average=0.0;  
    int i,med;  
    double temp;  
  
    for(i=0;i<n;i++)  
    {  
        average+=d[i];  
    }  
    average/=n;  
  
    med=d[0];  
    for(i=1;i<n;i++)  
        if(abs(med-average) > abs(d[i]-average))
```

```

        med=d[i];

    return med;
}

int findANumber( int (*func)(int d[], int s), int data[], int size)
{
    return (*func)(data,size);
}

int main()
{
    int data[10] = { 113, 345, 23, 75, 923, 634, 632, 134, 232, 98 };
    int num;

    num = findANumber(maximum, data, 10);
    printf("The maximum is %d.\n", num);
    num = findANumber(minimum, data, 10);
    printf("The minimum is %d.\n", num);
    num = findANumber(median, data, 10);
    printf("The median is %d.\n", num);

    return ;
}

```

我們也可以延伸此做法，設計宣告一個指向多個函式的指標陣列：

```

void (*funcs[])(void) = {insert, delete, update};

...

(*funcs[i])(); //呼叫第i個函式

```

18.7 結構體的彈性陣列成員

C99開始提供一個新的功能，允許我們為結構體設計彈性的陣列成員(Flexible Array Member)[]也就是未定義大小的陣列，但必須為所有成員的最後一個，且只能有一個。請參考下面的程式：

[flexibleMember.c](#)

```

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

struct vstring
{

```

```
int len;  
char chars[];  
// char *chars;  
};  
  
int main()  
{  
    struct vstring *str = malloc(sizeof(struct vstring)+10);  
    str->len=10;  
  
    return ;  
}
```

From:

<http://junwu.nptu.edu.tw/dokuwiki/> - Jun Wu的教學網站 國立屏東大學資訊工程學系

Permanent link:

<http://junwu.nptu.edu.tw/dokuwiki/doku.php?id=c:advancedpoint>

Last update: **2016/05/28 20:30**

