S&T2024

Computer Programming
(Part 2 – Advanced C Programming Language)

Chapter 5

Lecturer
A/Prof Tay Seng Chuan

E-mail: pvotaysc@nus.edu.sg

Office of the Provost
National University of Singapore

Ground Rules

- Switch off your handphone and pager
- No talking while lecture is going on
- No gossiping while the lecture is going on
- Raise your hand if you have question to ask
- Be on time for lecture
- Be on time to come back from the recess break to continue the lecture
- Bring your printed lecture notes to lecture or use a laptop for the e-copy. Do not use handphone to read the pdf file.
- Do not open any app, except this pdf file and an editing tool (to take notes).

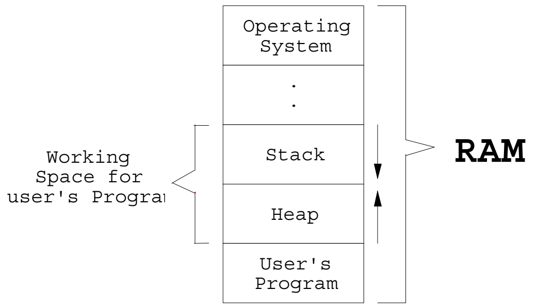
Chapter 5

Dynamic Memory Allocation

1 Introduction

- Some memory storage requirement cannot be determined at compilation time, such as the keyboard input and I/O requests.
- The storage is allocated dynamically at runtime. A program area called the heap is used to allocate memory dynamically.
- The heap is kept separate from the stack. It's possible, however, for the heap and stack to share the same memory segment.

3



A Simplified Semantic View of Memory

4

- A program that uses a large amount of memory for the stack may have a small amount of memory available for the heap and vice versa.
- You need to ensure that the storage is successfully allocated to store the data and to have some exception handling logic for the unsuccessful storage allocation.
- The heap is controlled by a heap manager, which allocates and deallocates (or returns) memory. User programs interface to the heap via C library calls.

2 C Functions for Dynamic Memory Allocation

- Bytes of memory are allocated in user-defined "chunks" (sometimes called objects) with
- **malloc()**
- **calloc()**
- **realloc()**
- **free()** relinquishes memory

Routine	Meaning
char *malloc(size)	allocate storage
unsigned size;	for size bytes
char *calloc(n, size)	allocate and zero storage
unsigned n, size;	for n items of size bytes
char *realloc(pheap, newsize)	reallocate storage
char *pheap;	for old heap pointer pheap
unsigned newsize;	for newsize bytes
void free(pheap)	free storage
char *pheap;	for heap pointer pheap

- ***sizeof()*** determines the number of bytes in the structure and makes the code portable.
- ***malloc()*** returns a heap address, and the program casts its return value from <u>a pointer to a char</u> (a byte pointer) to <u>a pointer to a structure</u> (structure block). You should always check the return value from malloc() and the other library calls before you use the heap address. A NULL (defined in stdio.h) indicates that the heap manager was unable to allocate storage.

Program

```
/* AC3-1.C - allocate block on heap */
#include<stdio.h>
#include<stdlib.h>
 struct block
{
  int header;
  char data[1024];
};

main()
{
   struct block *p;
   p=(struct block*) malloc (sizeof(struct block));
    if (p == NULL)
      {
         printf("malloc can't allocate heap space.\n");
         exit (1);
      }
   else
      {
         printf("Memory allocation successful!\n");
         free(p);      /* deallocate the memory */
      }
   return 0;
}
```

Screen Output

```
Memory allocation successful!
```

calloc()

- calloc() is similar to malloc(), but the routine fills heap memory with zeros. calloc(), therefore, runs slightly slower than malloc().
- calloc() takes two arguments, which are the number of objects to be allocated, and the size of each object.

  E.g., p2=(struct block*) calloc(MAXENTRIES, sizeof(struct block));
- calloc() also returns a heap address.

9

Program

```
/* AC3-2.C - allocate MAXENTRIES blocks on heap */
#include<stdio.h>
#include<stdlib.h>
#define MAXENTRIES 5
struct block
{
  int header;
  char data[1024];
};
main()
{
  struct block *p1,*p2;
  p1=(struct block*) malloc(MAXENTRIES * sizeof(struct block));
  if (p1 == NULL)
    {
      printf("malloc can't allocate heap space.\n");
      exit (1);
    }
  else
    {
      printf("Memory allocation for 'p1' is successful!\n");
      free(p1); /* return memory to the system */
    }
  p2=(struct block*) calloc(MAXENTRIES,  sizeof(struct block));
  if (p2 == NULL)
    {
      printf("calloc can't allocate heap space.\n");
      exit (1);
    }
  else
    {
      printf("Memory allocation for 'p2' is successful!\n");
      free(p2); /* return memory to the system */
    }
  return 0;
}
```

Screen Output

```
Memory allocation for 'p1' is successful!
Memory allocation for 'p2' is successful!
```

After the call to malloc(), p1 points to the first of five consecutive structures (of type block) in memory.

Similarly, p2 points to the first of five structures after the call to calloc(). The storage that p2 points to is zero-filled. Note that we pass two arguments to calloc(), but only one to malloc().

Each call allocates the same amount of heap memory.

10

realloc()

- **realloc()** allows you to change the size of any object on the heap.
- The routine's first argument is a pointer to a heap address. Presumably, this pointer is initialized from a previous call to the heap manager. The second argument is the number of bytes (including the existing) to be reallocated.

E.g.,
```
if ((num=(int*) realloc(num, size*(j+1)))==NULL)
  {
     printf("realloc fails.\n");
     exit(1);
  }
```

- **realloc()** can increase or decrease an object's size in heap memory.
- **realloc()** also preserves data in memory. If the storage space is being increased and there's not enough contiguous space on the heap, realloc() returns a **different address** than the one that you pass as its first argument. This means realloc() may have to move data; hence, its execution time varies.
- Occasionally, realloc() returns the *same* heap address. So, you have to maintain a table of heap addresses. When you allocate memory from the heap manager using any of the library calls, you should always update your table with the new pointer, even though it may not have changed. Programs that don't do this are not reliable.

11

- The heap manager frees heap memory with the C library call **free()**. You call it with a heap address returned from a previous call to malloc(), calloc(), or realloc(). The following statements free a structure called block from the heap:

```
struct block *p;
p = (struct block *) malloc (sizeof(struct block));
if (p == (struct block *) NULL)
{
   printf("malloc can't allocate heap space\n");
   exit(1);
}

/* processing on the block */
          .
          .
free(p);    /* free the structure */
          .
          .
```

You must pass a pointer to the start of some previously allocated space to free(). Note that free() doesn't return any thing.

12

4 Building an Array on the Fly

Suppose you are told to write a C program to perform a set of statistical analysis on the fly, how do you use array in a program without knowing the sample size ?

The following program demonstrates the way to build an array of ***arbitrary size***.

Program

```
 /* AC3-3.c */
#include<stdio.h>
#include<stdlib.h>

main()
{
  int count,size,i,j;
  int *num, sum=0;
  float average;

  printf("Enter as many integer values as you want! \n");
  printf("I will build an array on the fly with them and compute the average\n");
  printf("and the number of occurrences of the numbers which are less than\n");
  printf("the average.\n");
  printf("Note : Any non-number means you are done.\n");
  size = sizeof (int);
  if ((num=(int*) malloc (size))==NULL)
  {
    printf("malloc fails.\n");
   exit(1);
  }
```

13

```
   j=0;

  while (scanf("%d%*c",&num[j])==1)
   {
      sum+=num[j];
      j++;

     /* enlarge the array */
     if ((num=(int*) realloc(num,size*(j+1))) == NULL)
     {
       printf("realloc fails.\n");
       exit(1);
     }
    }

   average=(float)sum/j;
   count=0;

   for (i=0;i<j;i++) if (num[i]<average) count++;
   printf("Average = %f\n",average);
   printf("No. of occurrences below average = %d\n", count);
   return 0;
 }
```

Screen Output

```
 Enter as many integer values as you want!
I will build an array on the fly with them and
compute the average
and the number of occurrences of the numbers
which are less than
the average.
Note : Any non-number means you are done.
2800
3500
2000
1800
2200
3800
1800
4000
x
 Average = 2737.500000
No. of occurrences below average = 4
```

14

- The previous program is flexible as the size of the array can vary at runtime.
- However, the program is not efficient as the realloc function is invoked in each iteration.

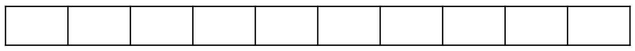
  The next program shows an example of dynamic memory allocation, but of a larger grain size. The program allocates a *contiguous 10 bytes* on the heap whenever the realloc function is called. Finally, the program *trims away* the storage which was allocated earlier but *not occupied*.

Program

```
/* AC3-4.C */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
main()
{
   char *first,*current;
   char this1;
   int  buffersize=10, increment=10, count=0;
   printf("\n Enter a string of any length and ");
   printf("\n press 'enter' key when you are done :");
   if ((first=current=malloc(buffersize))==NULL)
   {
     printf("malloc fails.\n");
     exit(1);
  }
```

```
    scanf("%c",&this1);
    while (this1 != '\n')
    {
      count++;
      *current=this1;
       if (count%increment==0)
         {
           buffersize += increment;
           if ((first = current = realloc(first,buffersize))==NULL)
            {
               printf("realloc fails.\n");
               exit(1);
            }
            current += count;
         }
      else
         current++;
       scanf("%c",&this1);
    }

   *current='\0';
    count++;
    first=realloc(first,count);
    printf(" input string = %s\nlength = %d\n", first,strlen(first));
     return 0;
  }
```



Screen Output

```
Enter a string of any length and press 'enter' key when you are done :C is so powerful that a lot of scientists and engineers have switched from other languages to it.
input string = C is so powerful that a lot of scientists and engineers have switched from other languages to it.
length = 96
```

5 Arrays of Pointers

- C lets you create arrays of any type of elements.
- You can even create an array whose elements are pointers.
- For example, to create an array of 10 pointers, in which each item is a pointer to a float, simply declare the following :

```
float  *array_name[10];
```

- The * preceding the array name in this declaration tells the compiler that the array is an *array of pointers*; therefore, *each element holds an address (pronounced as **where**)*.

- The float signifies that all pointers will point to float variables.

17

You can use this technique for speeding up some sorting programs. Suppose we are going to sort student records in the ascending order of matrix number where each record is defined as follows :

```
struct student
{
      char  matrix[10];
      char  dob[10];
      char  sex;
      char  subject[28][6];
};
```

Remember that for sorting data, we have to interchange their contents if they are not in order:

```
 int temp, i, j;
   if (i > j)
   {
      temp = i;
      i = j;
     j = temp;
   }
```

It incurs 3 assignment instructions for each interchange. Now in sorting our student records, each interchange involves

3 * (1+1+1+28) = 93 assignment instructions.

This is too expensive in terms of CPU time !!!

18

In the next program, we will use selection sort to sort an array of pointers, pointing to student records. Take note that if the matriculation numbers of two records are not in order, we will only interchange the pointers pointing to the records. The contents of the records remain unchanged.

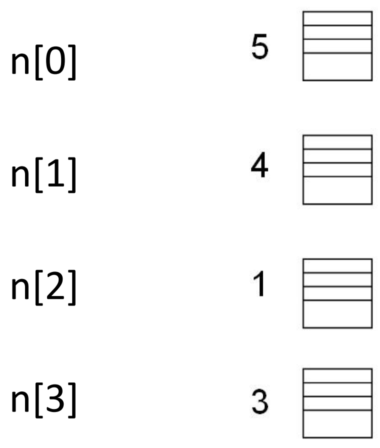
Input File (students.inf)

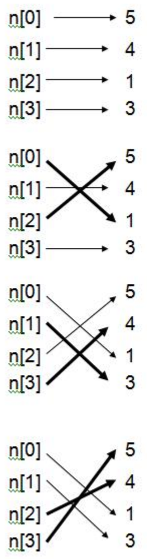
```
 942343D02 01/06/75 M CP111 CM101 GM101        . . .
946785U03 03/02/74 F CP112 PC101 MA101         . . .
945786V04 16/12/73 M PC111 CZ101 MA101
947894P01 23/11/76 F CP111 BA123 GM101
945676Z02 23/09/75 M CP112 DB101 GM101
947983X02 11/03/73 M CP111 CM101 GM101
948797U03 23/12/74 F CP112 PC101 MA101
943664V04 13/08/73 M BA111 CZ101 MA101
944865P01 20/01/74 M CP111 BA123 GM102
944564Z02 30/09/75 M CP101 PC101 GM101
```

Selection Sort

Unsorted Array	1st iteration	2nd iteration	3rd iteration
5			
4			
1			
3			

Selection Sort Using Pointers:





strcmp

Function	Compares one string to another
Syntax	# include <string.h> int strcmp (const char **s1*, const char **s2*);
Remarks	strcmp performs an unsigned comparison of *s1* and *s2*, starting with the first character in each string and continuing with subsequent characters until the corresponding characters differ or until the end of the string is reached.
Returns value	strcmp returns a value that is < 0 if *s1* is less than *s2* == 0 if *s1* is the same as *s2* >0 if *s1* is greater than *s2*

Program

```
/* AC3-5.C */
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 40
struct student
{
   char matrix[10];
   char dob[10];
   char sex;
   char subject[3][6];  /* To make life easy, I have declared a smaller two dimensional array of characters which
                           is slightly different from the one mentioned earlier. But the actual sorting strategy used in
                           this program can be applied to any amount of subjects. */
  } *st[MAXSIZE];

int GetRecord(void);
void SortRecord(int);

main()
{
  int no_of_st,i;
  no_of_st = GetRecord();
  SortRecord(no_of_st);
  for (i=0;i<no_of_st;i++)
  {
    printf("Matrix No. : %s.\t\tDate of Birth : %s.  \tSex : %c.\n",
                st[i]->matrix,st[i]->dob,st[i]->sex);
    printf("Subjects : %10s%10s%10s.\n\n",  st[i]->subject[0],
                            st[i]->subject[1], st[i]->subject[2]);
  }
  return 0;
}
```

23

```
GetRecord()
{
  int count=0,i;
  FILE *indata;
  indata=fopen("students.inf","r");

  do
  {
     st[count]=(struct student*)malloc(sizeof (struct student));
     fscanf(indata,"%s%s%*c%c%s%s%s", st[count]->matrix,
            st[count]->dob, &st[count]->sex,st[count]->subject[0],
            st[count]->subject[1], st[count]->subject[2] );
    count++;
  } while (!feof(indata));

  fclose(indata);
  return count;
}
```

```
void SortRecord(int n)        /*  Selection sort s used  */
{
  int i,j,min;
  struct student *temp;

  for (i=0;i<n;i++)
  {
    min=i;
    for (j=i+1;j<n;j++)  /* find the student with the smallest
                                matrix No. */
     if (strcmp(st[j]->matrix,st[min]->matrix)<0  min=j;
     if (min!=i)
       {
         temp=st[min];      /* only 3 interchanges */
         st[min]=st[i];
         st[i]=temp;
       }
   }
}
```

24

st[0]	matric[10]	dob	sex	sub[0]	sub[1]	sub[2]
st[1]	matric[10]	dob	sex	sub[0]	sub[1]	sub[2]
st[2]	matric[10]	dob	sex	sub[0]	sub[1]	sub[2]
st[3]	matric[10]	dob	sex	sub[0]	sub[1]	sub[2]
:						
:						
st[39]						

:						
st[i]						
:	945786V04					
:						
:						
:	943664V04					
st[n-1]						

25

Screen Output

```
 Matrix No. : 942343D02.        Date of Birth : 01/06/75.        Sex : M.
Subjects :    CP111   CM101   GM101.
 Matrix No. : 943664V04.        Date of Birth : 13/08/73.        Sex : M.
Subjects :    BA111   CZ101   MA101.
 Matrix No. : 944564Z02.        Date of Birth : 30/09/75.        Sex : M.
Subjects :    CP101   PC101   GM101.
 Matrix No. : 944865P01.        Date of Birth : 20/01/74.        Sex : M.
Subjects :    CP111   BA123   GM102.
 Matrix No. : 945676Z02.        Date of Birth : 23/09/75.        Sex : M.
Subjects :    CP112   DB101   GM101.
 Matrix No. : 945786V04.        Date of Birth : 16/12/73.        Sex : M.
Subjects :    PC111   CZ101   MA101.
 Matrix No. : 946785U03.        Date of Birth : 03/02/74.        Sex : F.
Subjects :    CP112   PC101   MA101.
 Matrix No. : 947894P01.        Date of Birth : 23/11/76.        Sex : F.
Subjects :    CP111   BA123   GM101.
 Matrix No. : 947983X02.        Date of Birth : 11/03/73.        Sex : M.
Subjects :    CP111   CM101   GM101.
 Matrix No. : 948797U03.        Date of Birth : 23/12/74.        Sex : F.
Subjects :    CP112   PC101   MA101.
```

26