Kỹ Thuật Lập Trình

Khang Q.H. Vo (*M.Sc.*)

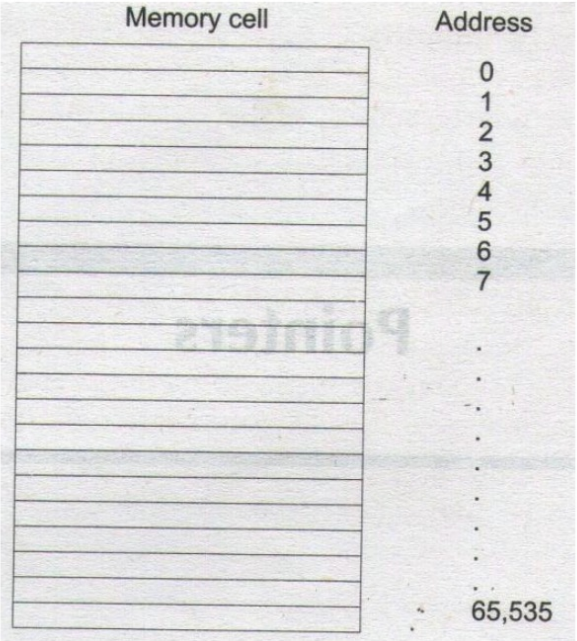
FIT - IUH

CON TRỎ - Pointers

NỘI DUNG

1. Address in C (Lưu trữ dữ liệu trong RAM)
2. Sơ đồ bộ nhớ (Memory Map of a program)
3. Giải thích con trỏ là gì?
4. Các phép toán trên con trỏ
5. Truyền tham số là con trỏ
6. Con trỏ và mảng 1 chiều
7. Cấp phát động

1. Address in C



Memory Organization

Consider the following statement, $int\ a\ =\ 179$; this statement instructs the system to find a location for the integer variable 'a' and puts the value 179 in that location.

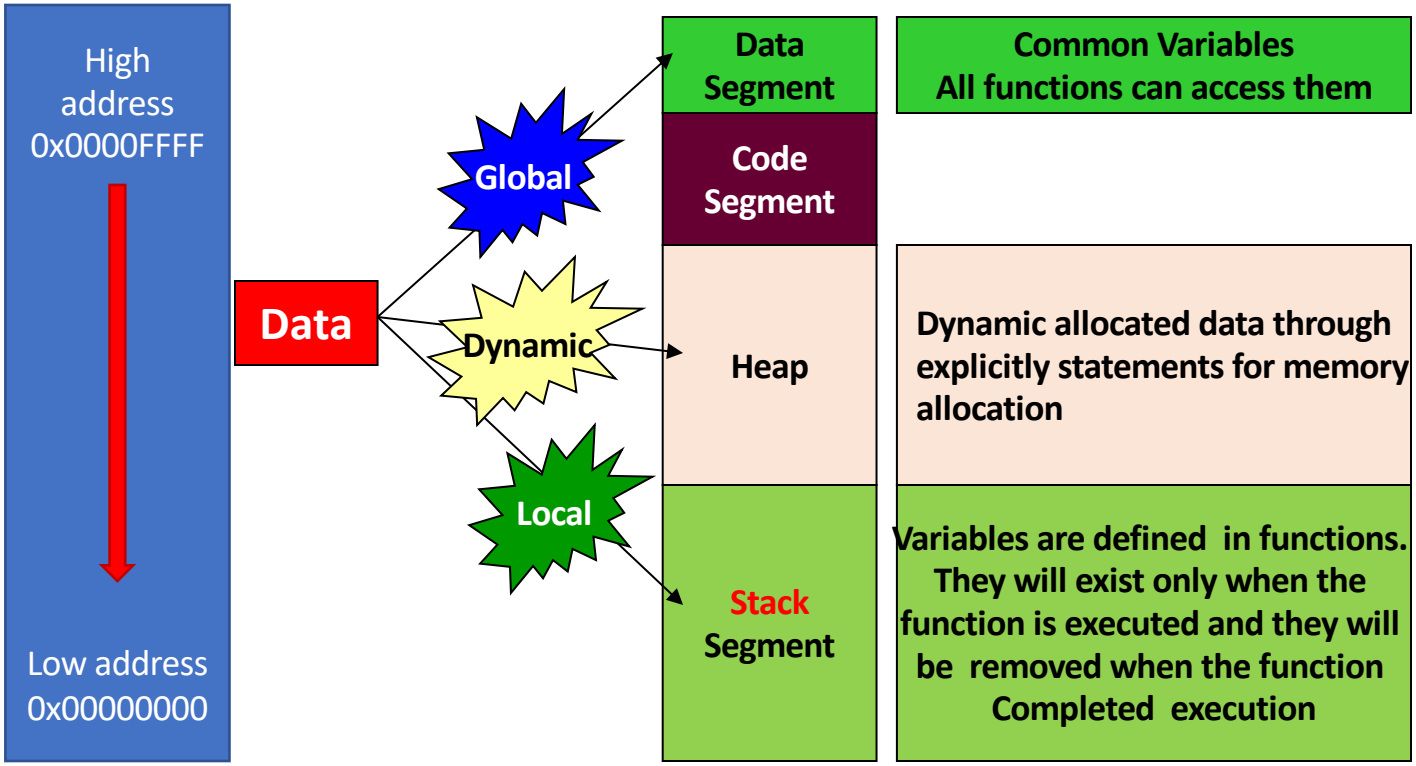
Variable → a
Value → 179
Address → 5000

Address in C - Example

```c
#include <stdio.h>
int main()
{
  int var = 5;
  printf("var: %d\n", var);

  // Notice the use of & before var
  printf("address of var: %p", &var);
  return 0;
}
```

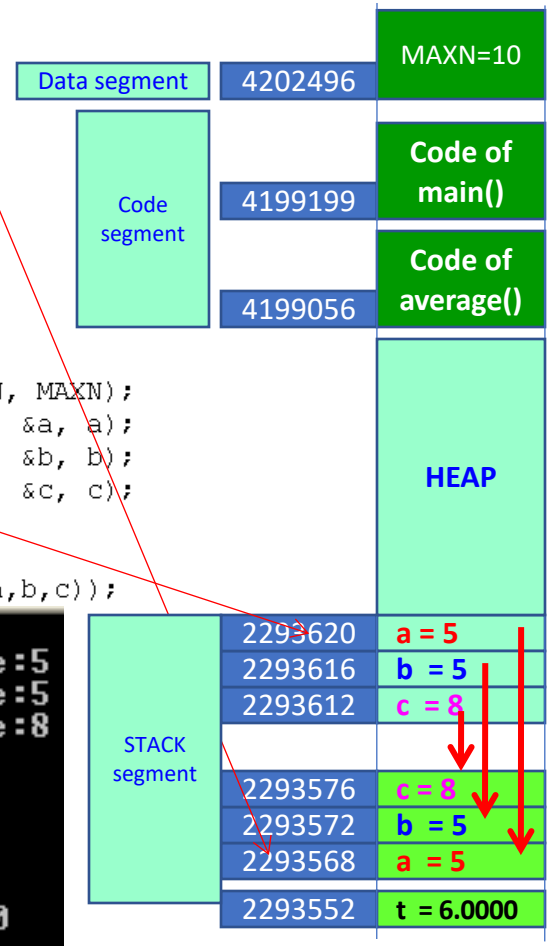
Where can we put program's data?



2. Memory Map of a program

```c
#include <stdio.h>
int MAXN=10;
double average (int a, int b, int c)
{  printf("Arg. a, address:%u, value:%d\n", &a, a);
   printf("Arg. b, address:%u, value:%d\n", &b, b);
   printf("Arg. c, address:%u, value:%d\n", &c, c);
   double t = (a+b+c)/3.0;
   printf("Var. t, address:%u, value:%lf\n", &t, t);
   return t;
}
int main()
{  int a= 5, b=5, c=8;
   printf("Var. MAXN, address:%u, value:%d\n", &MAXN, MAXN);
   printf("In main, var. a, address:%u, value:%d\n", &a, a);
   printf("In main, var. b, address:%u, value:%d\n", &b, b);
   printf("In main, var. c, address:%u, value:%d\n", &c, c);
   printf("Add. of main():%u\n", &main);
   printf("Add. of average(...):%u\n", &average);
   printf("Result returned to main: %lf\n",average(a,b,c));
}
```

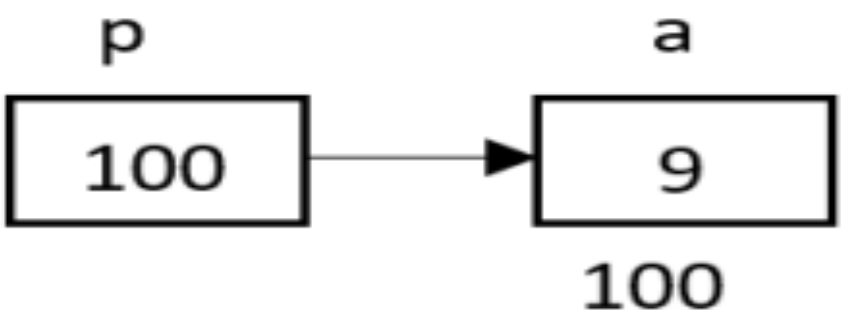
```
Var. MAXN, address:4202496, value:10
In main, var. a, address:2293620, value:5
In main, var. b, address:2293616, value:5
In main, var. c, address:2293612, value:8
Add. of main():4199199
Add. of average(...):4199056
Arg. a, address:2293568, value:5
Arg. b, address:2293572, value:5
Arg. c, address:2293576, value:8
Var. t, address:2293552, value:6.000000
Result returned to main: 6.000000
```



3. What is a Pointer?

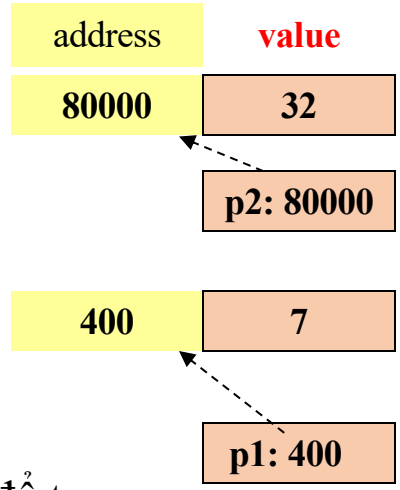
- Con trỏ là một biến, chứa địa chỉ của một biến khác.
- Xét đoạn code:

```
int a = 9;
int *p = &a;
```



3. What is a Pointer?

- Con trỏ là một biến, chứa địa chỉ của một vị trí bộ nhớ của một biến khác



- Nếu một con trỏ chứa địa chỉ của biến khác, được gọi là trỏ đến biến.
- Con trỏ cung cấp một phương thức gián tiếp để truy cập giá trị của biến.
- Con trỏ có thể trỏ đến các biến của các loại dữ liệu cơ bản khác như int, char hoặc double hoặc tổng hợp dữ liệu như mảng hoặc cấu trúc.

4. Pointer Variables

Syntax: **dataType *name;**

Examples:

```
int *pI;
double* pD;
char *pC;
```

Con trỏ có kiểu khác nhau thì có gì khác khau ?

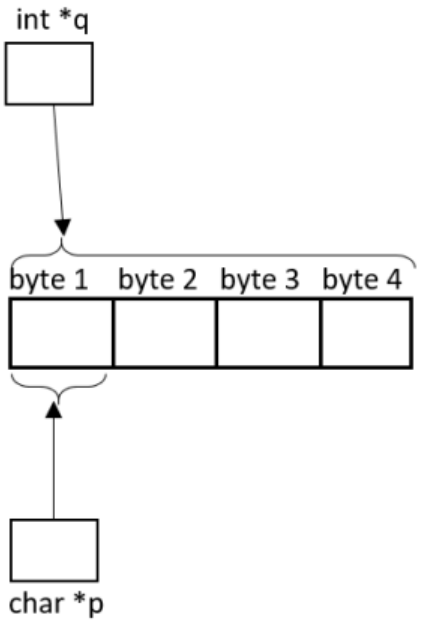
- Về bản chất, tất cả con trỏ dù là **char*** hay **int*** đều là số nguyên và kích thước của chúng đều bằng kiểu **int**.

```
1  //với hệ điều hành 32 bit
2  sizeof(char*) == sizeof(int*) ==
3  sizeof(int) == sizeof(void*) == 4
```

Con trỏ có kiểu khác nhau thì có gì khác khau ?

- Sự khác nhau nằm ở số byte khi truy cập vùng nhớ
- Xét đoạn code:

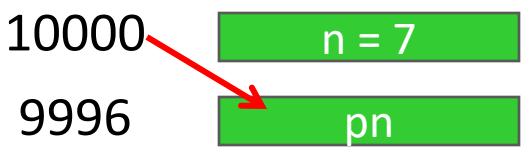
```
1  int n = 123456;
2  char *p = (char*)&n;
3  int *q = &n;
4  printf("*p = %d\n", *p);
5  printf("*q = %d\n", *q);
```

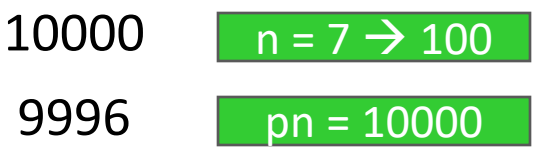


- p và q đều cùng trỏ đến byte đầu tiên của biến n.
- Tuy nhiên, khi truy cập đến vùng nhớ thông qua con trỏ, con trỏ **char *p** chỉ đọc **1 byte**, còn con trỏ **int *q** sẽ đọc **4 byte**

5. Pointer Operators

How to	Operator	Example
Get address of a variable and assign it to a pointer	&	int n= 7; int* pn = &n;
Access indirectly value of a data through it's pointer	*	*pn =100;





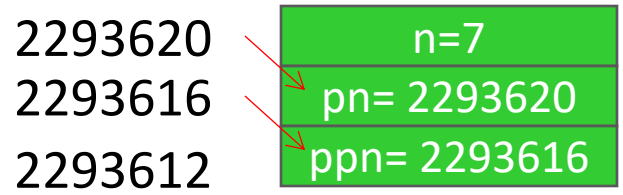
int n= 7;
int* pn = &n;

pn = &n; → pn=10000

*pn = 100;

*pn = 100; → Value at [10000] =100

Pointer Operators…



n → int → pn stores address of n
→ **pn: int***
pn → int* → ppn stores address of pn
→ ppn: (int*)* → **ppn: int****

```c
#include <stdio.h>
int main()
{   int n=7;
    int*pn = &n;
    int**ppn = &pn;
    printf("Variable n  : addr: %u, value:%d\n", &n, n);
    printf("Variable pn : addr: %u, value:%u\n", &pn, pn);
    printf("Variable ppn: addr: %u, value:%u\n", &ppn, ppn);
    getchar();
    return 0;
}
```

```
K:\GiangDay\FU\OOP\BaiTap\pointer_demo1.exe
Variable n  : addr: 2293620, value:7
Variable pn : addr: 2293616, value:2293620
Variable ppn: addr: 2293612, value:2293616
```

Pointer Operators… Walkthrough

100	n=7 → 54
96	m=6 → -30
92	pn=100
88	pm= 96

```
#include <stdio.h>
int main()
{   int n=7, m=6;
    int*pn = &n;
    int*pm = &m;
    *pn = 2*(*pm) + m*n;
    *pm += 3*m - (*pn);
    printf("m= %d, n=%d\n", m, n);
    getchar();
    return 0;
}
```

K:\GiangDay\FU\OOP\BaiTap\pointer_demo2.exe

```
m= -30, n=54
```

pn = 2(*pm) + m*n;
Value at 100 = 2*(value at 96) + m *n
Value at 100 = 2*6 + 6 *7
Value at 100 = 12 + 42 = 54

*pm += 3*m - (*pn);
Value at 96 += 3*6 – value at 100
Value at 96 += 3*6 – 54
Value at 96 += 18 – 54
Value at 96 += (-36)
Value at 96 = 6+ (-36) = -30

Walkthroughs: Do yourself

```c
#include <stdio.h>
int main()
{   int n=7, m=6;
    int*pn = &n;
    int*pm = &m;
    *pn = *pm + 2*m-3*n;
    *pm -= *pn;
    printf("%d", m+n);
    getchar();
    return 0;
}
```

```
K:\GiangDay\FU\OOP\BaiTap\poi
6_
```

```c
#include <stdio.h>
int main()
{   double x= 3.2, y= 5.1;
    double* p1= &x;
    double* p2= &y;
    *p1 += 3 - 2*(*p2);
    *p2 -= 3*(*p1);
    printf("%lf", x+y);
    getchar();
    return 0;
}
```

```
K:\GiangDay\FU\OOP\BaiTap\pointer
13.100000_
```

```c
#include <stdio.h>
int main()
{   char c1='A', c2= 'F';
    char* p1= &c1;
    char* p2= &c2;
    *p1 += 3;
    *p2 -=5;
    printf("%d", c1-c2);
    getchar();
    return 0;
}
```

```
K:\GiangDay\FU\OOP\BaiTap\pointe
3_
```

int n=7,m=8;

int* p1= &n, *p2=&m;

*p1 +=12-m+ (*p2);

p2 = m + n- 2(*p1);

printf("%d", m+n);

What is the output?

int n=7, m=8;

int* p1= &n, *p2=&m;

p1 +=5 + 3(*p2) –n ;

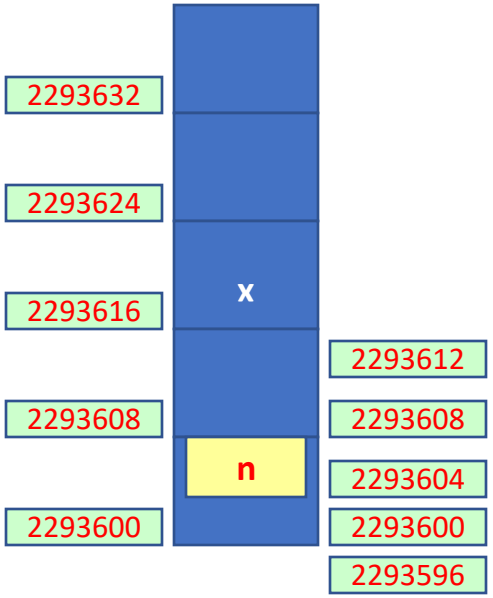
p2 = 5(*p1) – 4*m + 2*n;

What are values of m and n?

6. Pointer Arithmetic Operators: +, -, ++, --

```c
#include <stdio.h>
int main()
{   double x = 0.5;
    double* pD = &x;
    int i;
    for (i= -2; i<=2; i++) printf("%u, ", pD+i);
    printf("\n");
    int n= 3;
    int *pI = &n;
    for (i= -2; i<=2; i++) printf("%u, ", pI+i);
    printf("\n");
    pI++;
    printf("%u\n", pI);
    pI--;
    printf("%u\n ", pI);
    getchar();
    return 0;
}
```

Pointer +i → Pointer + (i*sizeof(baseType))




```
2293600, 2293608, 2293616, 2293624, 2293632,
2293596, 2293600, 2293604, 2293608, 2293612,
2293608
2293604
```

Pointer Arithmetic Operators : Accessing the neighbor

Copy and paste, run the program, explain the result.

```
/* file pointer_demo4.c */
#include <stdio.h>
int main()
{  int n2= 10;
   int n1= 6;
   int n0= 5;
   printf("n2=%d, n1=%d, n0=%d\n", n2, n1, n0);
   int* p = &n1;
   *p=9;
   p++;
   *p=15;
   p--;
   p--;
   *p=-3;
   printf("n2=%d, n1=%d, n0=%d\n", n2, n1, n0);
   return 0;
}
```

Exercises

double *p;
Suppose that a double occupies the memory block of 8 bytes and
p stores the value of 1200.
What are the result of the following expression? p+8 p-3 p++

long*p;
Suppose that a long number occupies the memory block of 4 bytes
And p stores the value of 1000.
What are the result of the following expression? p+8 p-3 p++

char*p;
Suppose that a character occupies the memory block of 1 byte
and p stores the value of 207000.
What are the result of the following expression? p+8 p-3 p++

Pointer +i → Pointer + (i*sizeof(baseType))

7. Pointers as Parameters of a Function

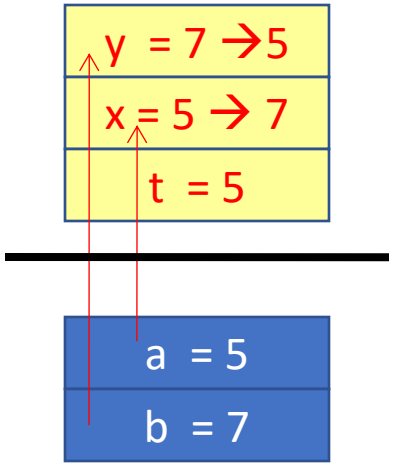
- C uses by-value parameters only → A function can not modify values of arguments.
- To modify values of arguments, pointers as parameters of a function are used.

7. Pointers as Parameters of a Function

- C passes arguments to parameters by values only → C functions can not modify outside data.

```
/* file pointer_demo5.c */
#include <stdio.h>
/* swap 2 integers */
void swap1 ( int x, int y)
{ int t= x;
  x = y;
  y = t;
}
int main()
{   int a= 5, b=7;
    printf("a=%d, b=%d\n", a, b);
    swap1(a,b);
    printf("a=%d, b=%d\n", a, b);
    getchar();
    return 0;
}
```

```
K:\GiangDay\FU\OOP\BaiTap\pointer_demo6.exe
a=5, b=7
a=5, b=7
```

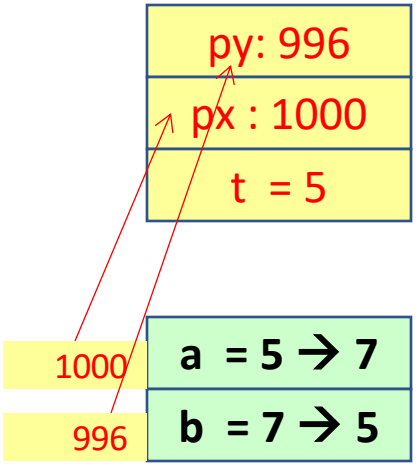


7. Pointer as Parameters of a Function …

Use pointer arguments, we can modify outside values.

```
/* file pointer_demo5.c */
#include <stdio.h>
/* swap 2 integers at pointers */
void swap2 (int* px, int* py)
{ int t= *px;/* t= value at px */
  *px = *py; /* value at px = value at py */
  *py = t;   /* value at py = t */
}
int main()
{  int a= 5, b=7;
   printf("a=%d, b=%d\n", a, b);
   swap2( &a, &b );
   printf("a=%d, b=%d\n", a, b);
   getchar();
   return 0;
}
```

```
K:\GiangDay\FU\OOP\BaiTap\pointer_demo6.exe
a=5, b=7
a=7, b=5
```

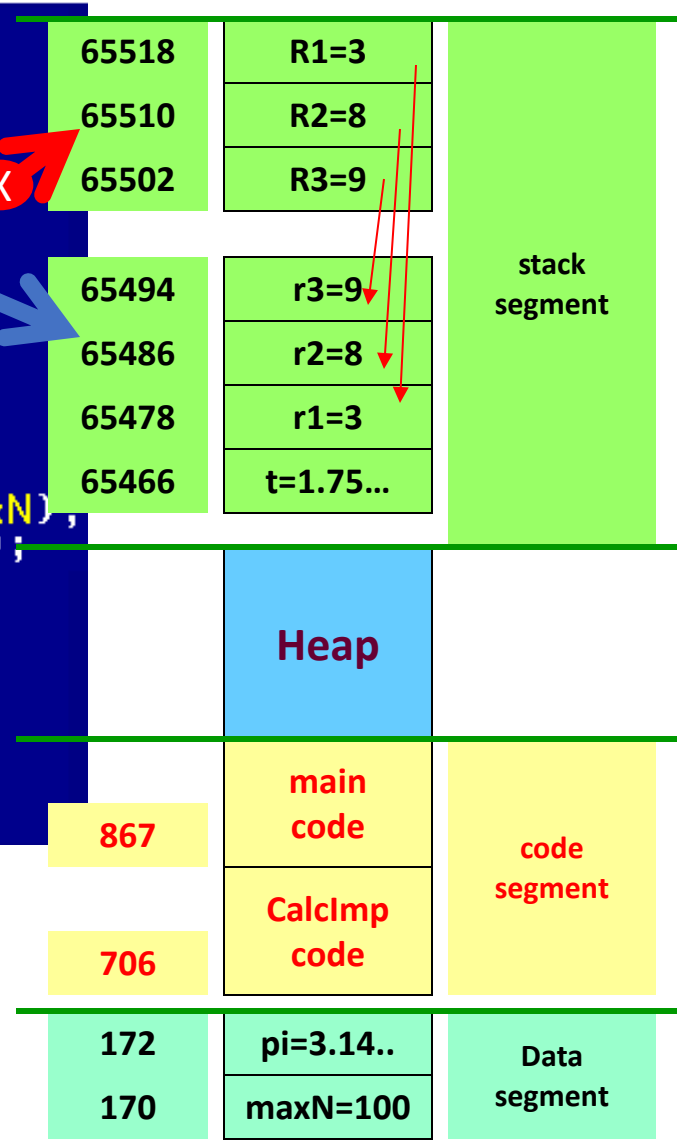


Functions with pointers as parameters

Review: Pass by-value Parameters

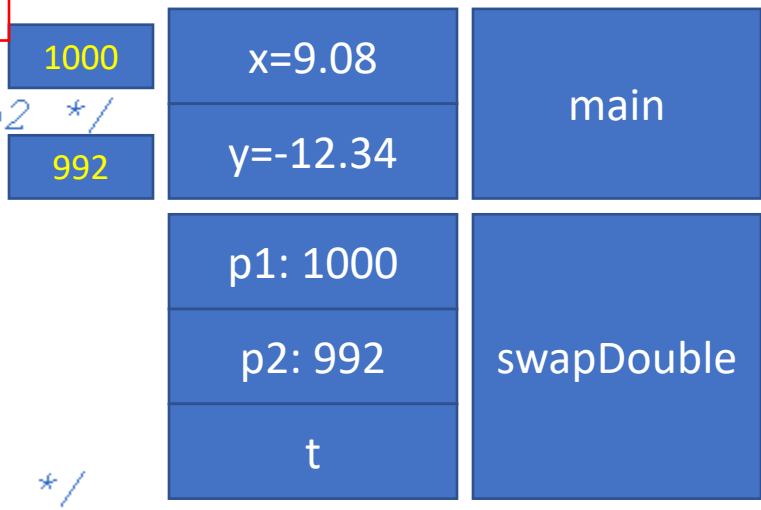
```
#include<stdio.h>
#include<conio.h>
int maxN=100;
double pi=3.141592;
double CalcImp(double r1, double r2, double r3)
{   double t=1/(1/r1 + 1/r2 + 1/r3);
    printf("r1    Addr:%u,value:%lf\n", &r1,r1);
    printf("r2    Addr:%u,value:%lf\n", &r2,r2);
    printf("r3    Addr:%u,value:%lf\n", &r3,r3);
    printf("t     Addr:%u,value:%lf\n", &t  ,t);
   return t;
}
int  main()
{ double R1=3, R2=8, R3=9;
  printf("maxN Addr:%u,  value:%d \n", &maxN,maxN);
  printf("pi   Addr:%u,  value:%lf\n", &pi   ,pi);
  printf("R1   Addr:%u,value:%lf\n", &R1   ,R1);
  printf("R2   Addr:%u,value:%lf\n", &R2   ,R2);
  printf("R3   Addr:%u,value:%lf\n", &R3   ,R3);
  printf("main    addr:%u\n", &main);
  printf("CalcImp addr:%u\n", &CalcImp);
  printf("Impedance: %lf",CalcImp(R1,R2,R3));
  getch();
}
```

```
maxN Addr:170,  value:100
pi   Addr:172,  value:3.141592
R1   Addr:65518,value:3.000000
R2   Addr:65510,value:8.000000
R3   Addr:65502,value:9.000000
main    addr:867
CalcImp addr:706
r1   Addr:65478,value:3.000000
r2   Addr:65486,value:8.000000
r3   Addr:65494,value:9.000000
t    Addr:65466,value:1.756098
Impedance: 1.756098
```



7. Pointers as parameters: Demo

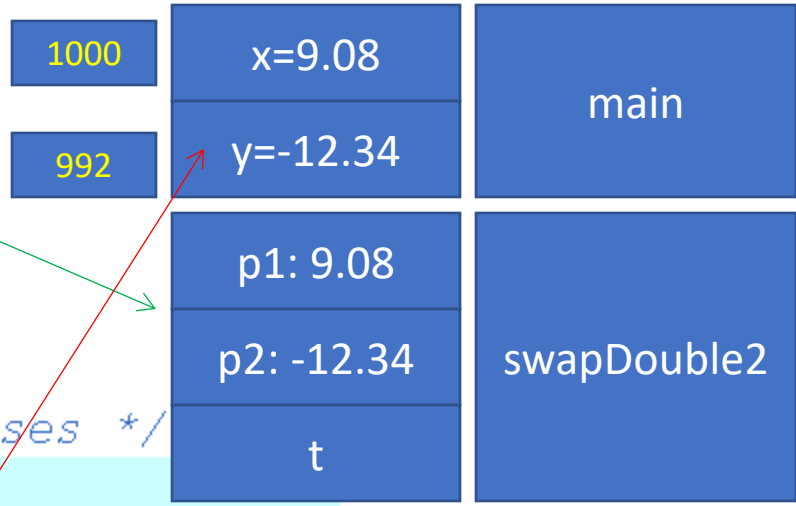
```
1  /* Accept 2 numbers, swap them, then print out them */
2  #include <stdio.h>
3  /* SWAPPING 2 DOUBLE NUMBERS AT ADDRESSES p1, p2 */
4  void swapDouble (double *p1, double *p2)
5  {   double t=*p1; /* t = value at p1 */
6      *p1= *p2; /* value at p1 = value at p2 */
7      *p2= t; /* value at p2 = t */
8  }
9  int main()
10 {   double x, y;
11     printf("Enter 2 real numbers:");
12     scanf("%lf%lf", &x, &y);
13     /* swaping 2 values at their addesses */
14     swapDouble2 (x, y);
15     printf("After swapping x=%lf, y=%lf\n", x, y);
16     fflush(stdin);
17     getchar();
18     return 0;
19 }
```

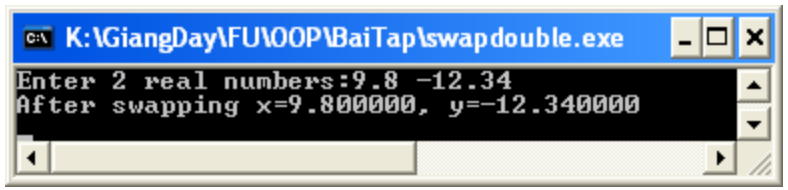


swapDouble(&x, &y);

7. Pointers as parameters: Demo

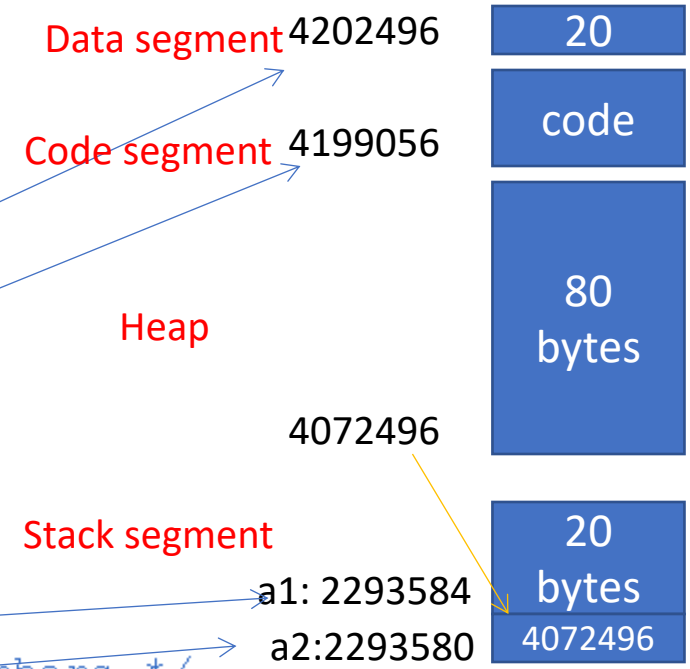
```
9  void swapDouble2 (double p1, double p2)
10 {   double t=p1;
11     p1= p2;
12     p2= t;
13 }
14
15 int main()
16 {   double x, y;
17     printf("Enter 2 real numbers:");
18     scanf("%lf%lf", &x, &y);
19     /* swaping 2 values at their addesses */
20     swapDouble2 (x, y);
21     printf("After swapping x=%lf, y=%lf\n", x, y);
22     fflush(stdin);
23     getchar();
24     return 0;
25 }
```



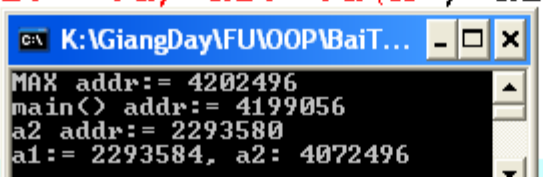


8. Pointers and Arrays

The name of the array is the address of the first element.

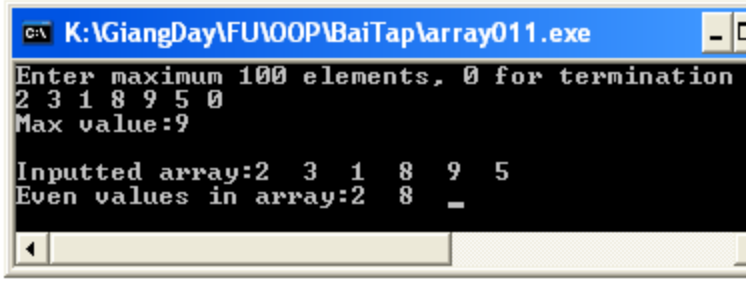



```c
#include <stdio.h>
#include <stdlib.h>
int MAX=20;
int main()
{   int a1[5]; /* static array of 5 int numbers */
    double *a2=NULL; /* dynamic array of double numbers */
    printf ("MAX addr:= %u\n", &MAX);
    printf ("main() addr:= %u\n", &main);
    /* allocate a mem. block for 10 double numbers */
    a2 = (double*)calloc(10,sizeof(double));
    printf ("a2 addr:= %u\n", &a2);
    printf ("a1:= %u, a2: %u\n", a1, a2);
    getchar();
    return 0;
}
```



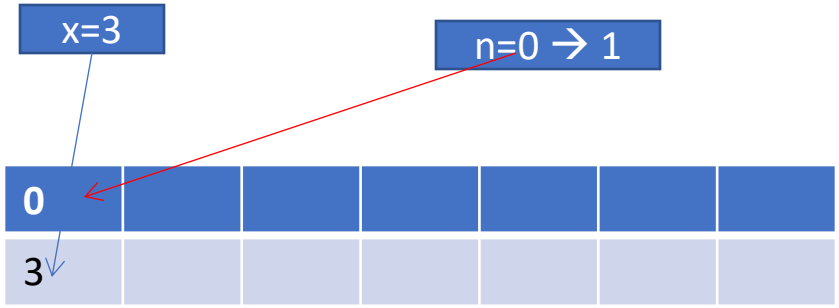
Array Function Parameter: Demo

```c
#include <stdio.h>
#define MAXN 100
/* Input an array, number of elements is stored at pn
   User will terminate inputting when 0 is entered.*/
void input(int*a, int *pn);
int max(int a[], int n);
void print (int* a, int n);
void printEven (int* a, int n);
int main()
{   int a[MAXN]; /* static array of 100 integers */
    int n; /* real used number of elements */
    int maxVal;
    input(a,&n);
    maxVal  = max (a,n);
    printf("Max value:%d\n", maxVal);
    printf("\nInputted array:");
    print(a,n);
    printf("\nEven values in array:");
    printEven(a,n);
    while (getchar()!='\n');getchar();
    return 0;
}
```

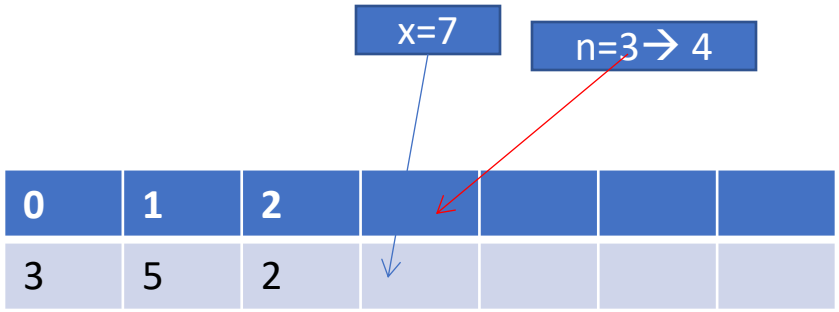


Array Function Parameter: Demo

```
24 void input(int*a, int *pn)
25 { *pn=0; /* reset the number of elements */
26   printf ("Enter maximum %d elements, 0 for termination\n", MAXN);
27   int x; /* inputted value */
28   do
29   { scanf("%d", &x);
30     if (x!=0) a[(*pn)++] = x;
31   }
32   while (x!=0 && *pn < MAXN);
33 }
```




```
34 int max(int a[], int n)
35 {
      /* Do yourself */
42 }
43 void print (int* a, int n)
44 { /* Do yourself */
47 }
48 void printEven (int* a, int n)
49 { /* Do yourself */
53 }
```



9. Dynamic Allocated Data

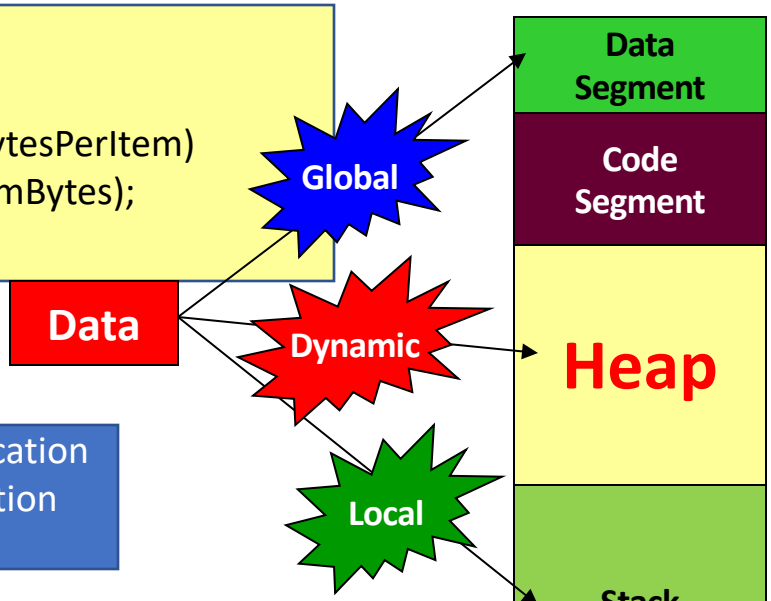
Stdlib.h

```
void*  malloc (size_t  numBytes) ;
void*  calloc  (size_t numberOfItem, size_t  bytesPerItem)
void* realloc (void* curPointer, size_t newNumBytes);
void  free(void* willBeDeletedPointer);
```

Examples

```
int* p = (int*)  malloc (sizeof (int)) ;
*p=2;
....
 free(p);
```

calloc: Contiguous Allocation
malloc: Memory allocation
realloc: Re allocation



size_t: Another name of the **int** type. It is used in case of memory allocation managing.
void is the general datatype which means that the data type is not determined yet. So, user must give an explicit casting when it is used.

Demo: Dynamic Allocated Data

```
/* file pointer_demo5.c */
#include <stdio.h>
#include <stdlib.h>
const int MAXN =100;
int main()
{
  int n;   int *p1;   int *p2;   int *p3;
  printf("Address of MAXN: %u\n", &MAXN);
  printf("Main function ia allocated at: %u\n", &main);
  printf("Address of n : %u\n", &n);
  printf("Address of p1: %u\n", &p1);
  printf("Address of p2: %u\n", &p2);
  printf("Address of p3: %u\n", &p3);
  p1 = (int*)malloc(sizeof(int));
  p2 = (int*)malloc(sizeof(int));
  p3 = (int*)malloc(sizeof(int));
  printf("Dynamic allocation (p1) at: %u\n", p1);
  printf("Dynamic allocation (p2) at: %u\n", p2);
  printf("Dynamic allocation (p3) at: %u\n", p3);
  free(p1);
  free(p2);
  return 0;
}
```

(1) Copy, past, compile and run the program.
(2) Draw the memory map.
(3) Show that where is data segment, code segment, stack segment and heap of the program.
(4) Give comment about the direction of dynamic memory allocation.

Dynamic Allocated Data- Do yourself

Use dynamic memory allocation. Develop a program that will accept two real numbers then sum of them, their difference, their product, and their quotient are printed out.

```
/* main() */
double *p1, *p2;
p1 = (double*) malloc ( sizeof(double));
p2 = (double*) malloc ( sizeof(double));
printf("p1, address: %u, value: %u\n", &p1, p1);
printf("p2, address: %u, value: %u\n", &p2, p2);
printf("Input 2 numbers:");
scanf( "%lf%lf", p1, p2);
printf("Sum: %lf\n", *p1 + *p2);
printf("Difference: %lf\n", *p1 - *p2);
printf("Product: %lf\n", *p1 * (*p2));
printf("Quotient: %lf\n", *p1 / *p2);
```

(1) Run this program
(2) Draw the memory map (stack, heap).

Dynamic Allocated Data- Do yourself

Write a C program using dynamic allocating memory to allow user entering two characters then the program will print out characters between these in ascending order.

<u>Example</u>:

Input: DA

Output:

A 65 81 41
B 66 82 42
C 67 83 43
D 68 84 44

```
- char* pc1, *pc2;
-Cấp bộ nhớ pc1, pc2;
- Nhập 2 ký  tự vào pc1, pc2
- Nếu (*pc1>*pc2) Hoán vị *pc1, *pc2;
-char c;
-For (c= *pc1; c<=*pc2; c++)
-     printf("%c,%4d,%4o%4X\n", c,c,c,c);
```

After the program executes, draw the memory map of the program.

Dynamic array

- Array is the simplest data structure for a group of elements which belong to the same data type.
- Each element in an array is identified by one or more index beginning from 0.
- Number of dimensions: Number of indexes are used to identify an element.
- Static arrays → Stack segment

```
Type a[MAXN];
```

- Dynamic array: Use pointer and allocate memory using functions

```
double *a = (double*)calloc(n, sizeof(double));
```

Dynamic array

- Accessing elements in an array:

1-D Array (a)	
Address	***Value***
&a[index]	a[index]
a+index	*(a+index)
Compiler determines the address of an element:	
a + index*sizeof(DataType)	

- **Common operations on arrays:**
  - Add an element
  - Search an element
  - Remove an element
  - Input
  - Output
  - Sort

- Base of algorithms on arrays: Traversing

Exercise- Do yourself

- Develop a C-program that helps user managing an 1-D array of real numbers(maximum of 100 elements) using the following simple menu:
- 1- Add a value
- 2- Search a value
- 3- Print out the array
- 4- Print out values in a range (minVal<=value<=maxVal, minVal and maxVal are inputted)
- 5- Print out the array in ascending order (positions of elements are preserved)
- Others- Quit

Summary

- Review the memory structure of a program
- Where can we put program's data?
- What are pointers?
- Pointer Declarations
- Where are pointers used?
- Pointer operators
  - Assign values to pointers
  - Access data through pointer
  - Explain pointer arithmetic
  - Explain pointer comparisons
- Pointers as parameters of a function
- Dynamic Allocated Data

Q&A