Slots 11-12
Pointers

Objectives

After studying this chapter, you should be able to:

- Understand where program's data can be putted
- Explain what are pointers
- Declare pointers in a program
- Discuss about where pointers can be used
- Understand operators on pointers
- Implement functions in which pointers are parameters
- Use build-in functions to allocate data dynamically

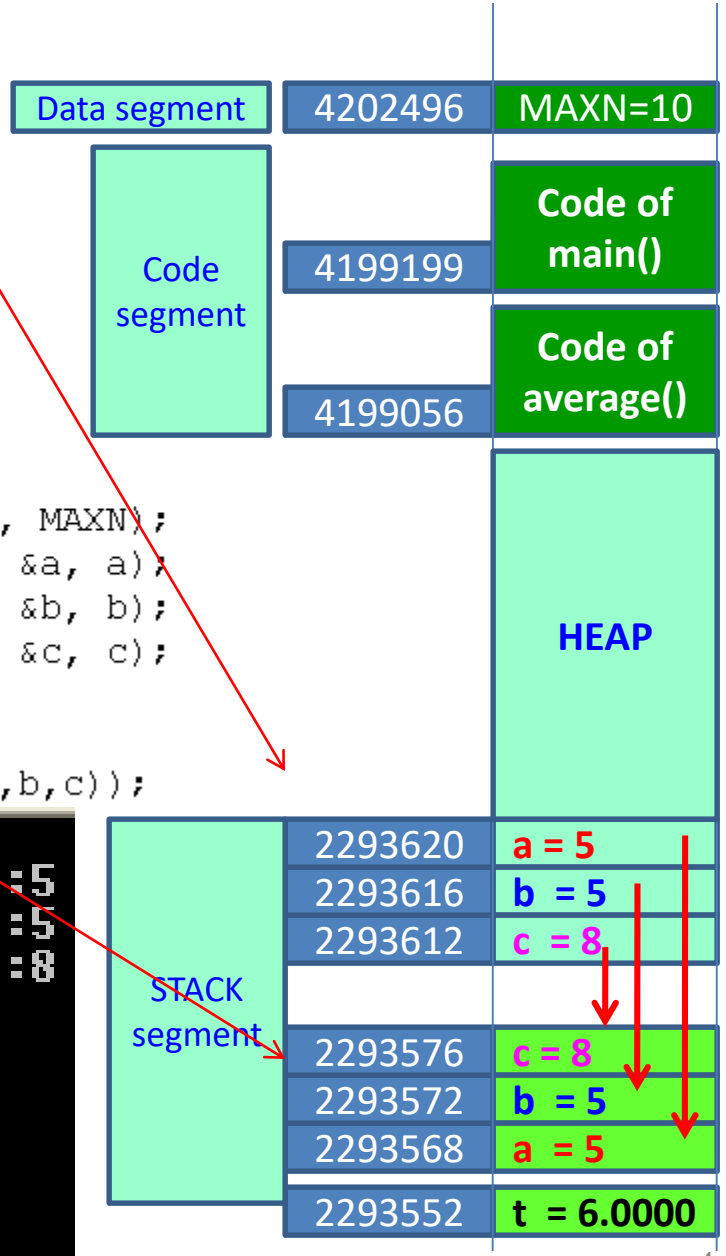
Contents

- 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

Memory Map of a program

```
#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
```

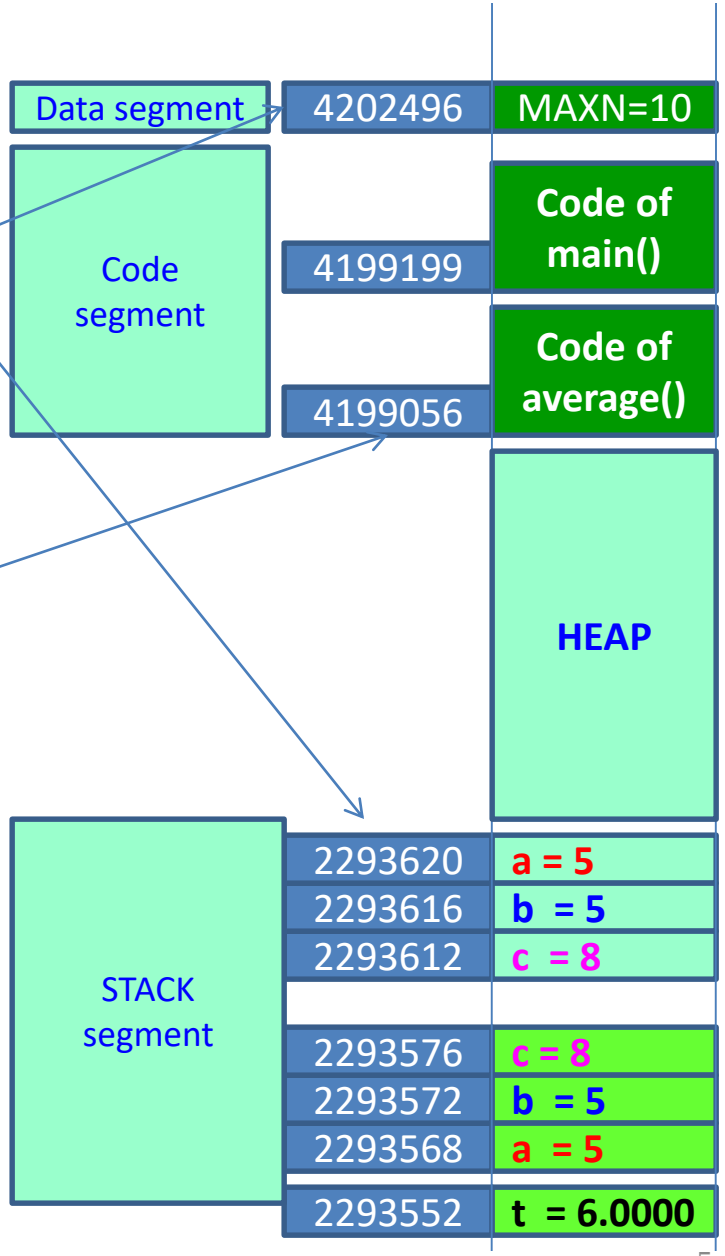


Questions

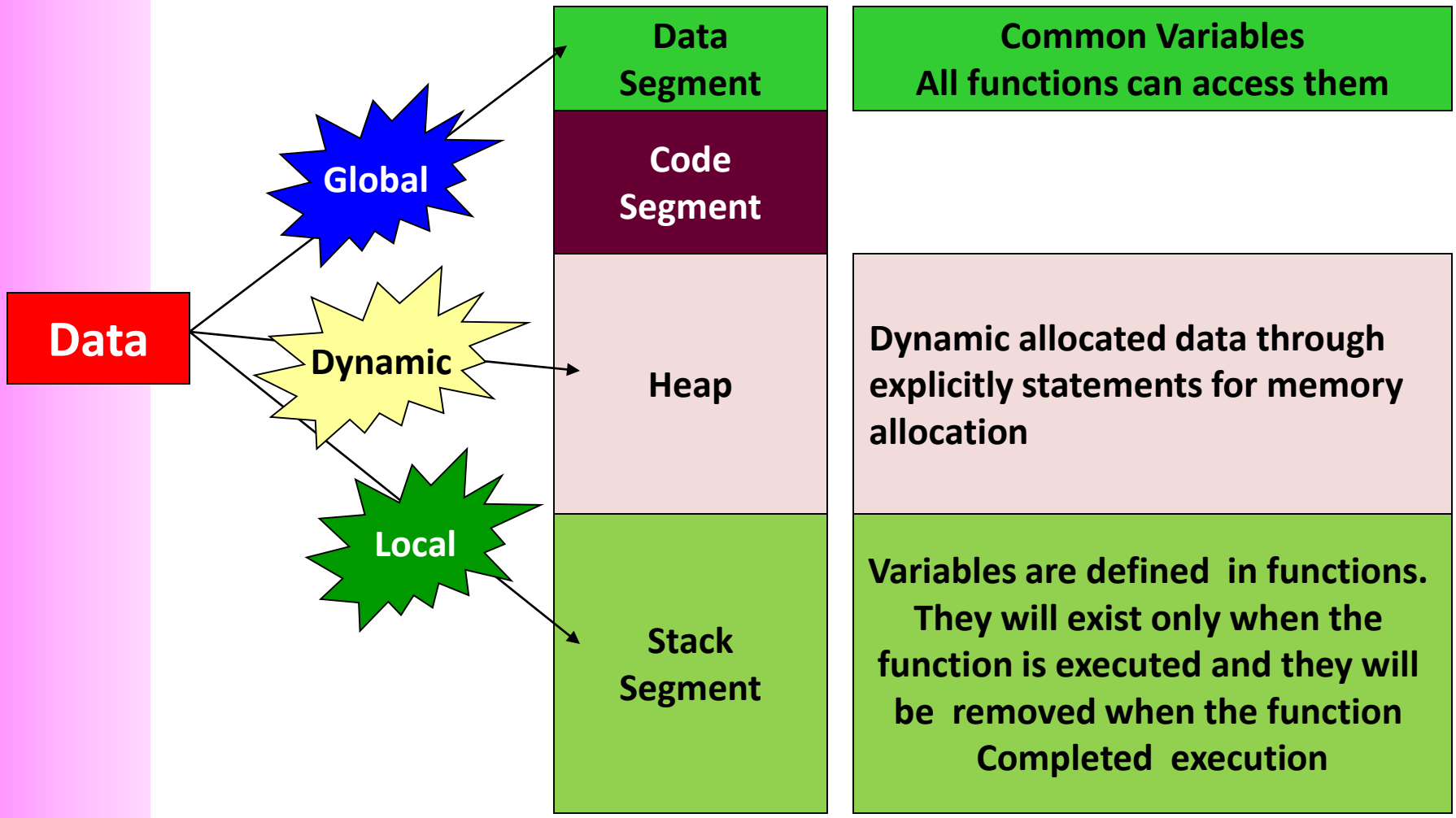
Address of a variable is a number. Can we assign this number to another variable then access data through the new variable?

Address of a function is a number. Can we assign this number to another variable then call this function through the new variable?

Yes. We can access a data through it's address and call a function through it's address also. POINTER is a way to satisfy these requirement.
In this chapter, pointers of variables are concerned only.

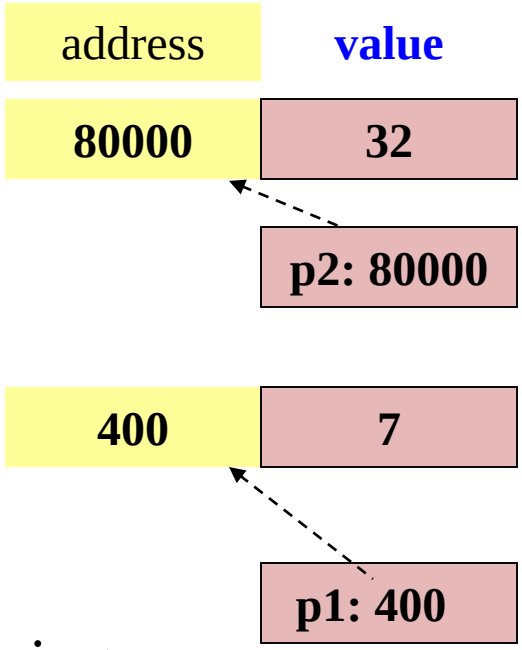


2- Where can we put program's data?



3- What is a Pointer?

- A pointer is a variable, which contains the address of a memory location of another variable
- If one variable contains the address of another variable, the first variable is said to point to the second variable



- A pointer provides an indirect method of accessing the value of a data item
- Pointers can point to variables of other fundamental data types like int, char, or double or data aggregates like arrays or structures

4- Pointer Variables

A pointer declaration consists of a base type and a variable name preceded by an *

Syntax: **dataType *name;**

The variable **name** will contain the address of a data belonging to the type **dataType**

Examples:

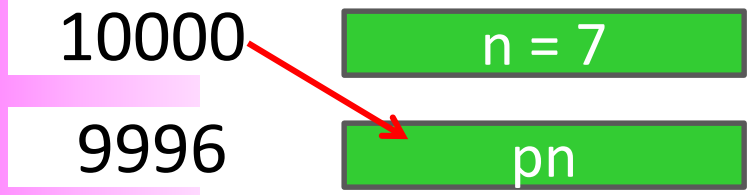
```
int *pI;
double* pD;
char *pC;
```

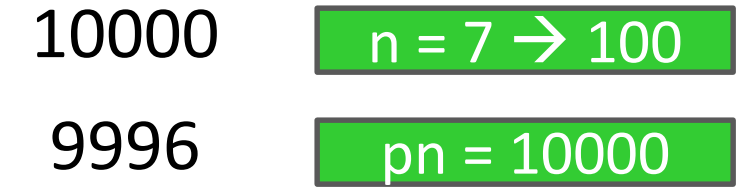
5- Where are pointers used?

- Some situations where pointers can be used are:
  - To modify outside arguments of a function
  - To return more than one value from a function (*It will be introduced in slots 18→24*)
  - To pass array and strings more conveniently from one function to another (*It will be introduced in slots 18→24*)
  - To manipulate arrays easily by moving pointers to them instead of moving the arrays itself (*It will be introduced in slots 18→24*)
  - To allocated memory and access it (Direct memory allocation) (*It will be introduced at the bottom of this lesson*)

6- 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;





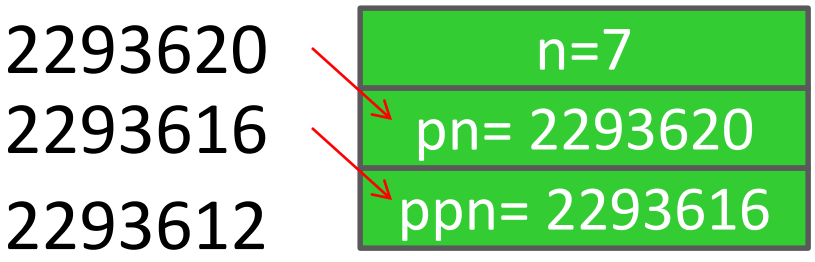
*pn = 100;

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

pn = &n; → pn=10000

*pn = 100; → Value at [10000] =100

Pointer Operators…

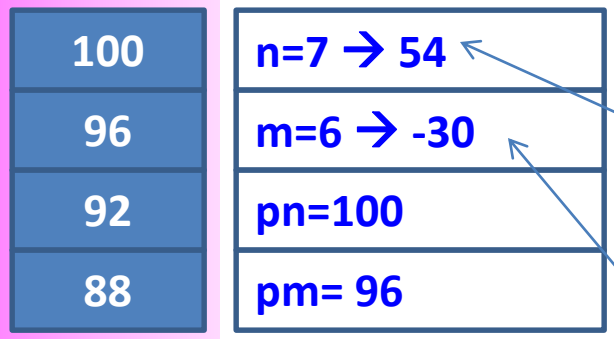


n → int → pn stores address of n
→ **pn: int***
pn → int* → ppn stores address of pn
→ ppn: (int*)* → **ppn: int****

```
#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




```
#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;
}
```

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?

Attention about Accessing Pointers

- Accessing data through pointers will manipulate on basic-data size.

→ Access int* → 4 bytes are affected.

→ Access char* → only 1 byte is affected.

→ Access double* → 8 bytes are affected.

→ Assign pointers which belong to different types are not allowed. → If needed, you must explicitly casting.

```
1 #include <stdio.h>
2 int main()
3 {   double x = 0.5;
4     double* pD = &x;
5     int * pI;
6     pI= pD;
7     getchar();
8     return 0;
9 }
```

Compiler | Resources | Compile Log | Debug | Find F

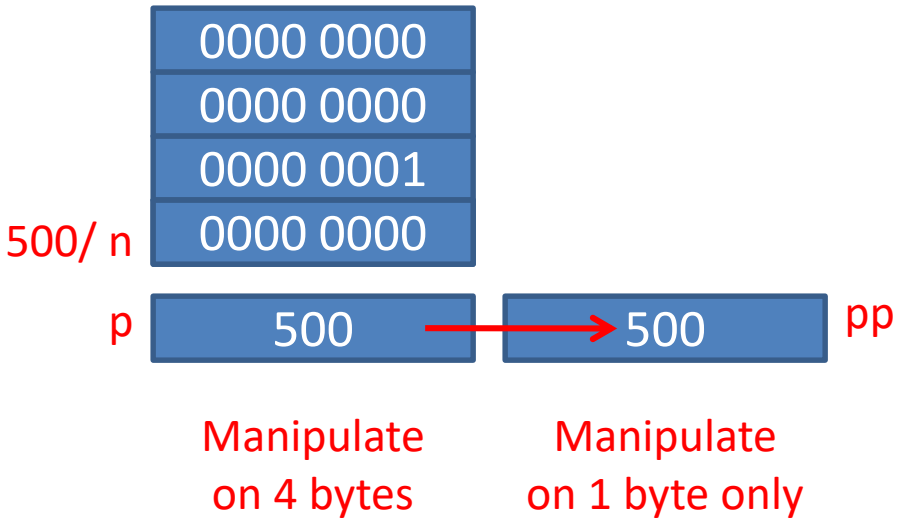
Line	File	Message
	K:\Gia...	In function `main':
6	K:\Gia...	[Warning] assignment from incompatible pointer type

Attention…Pointers: Explicit Casting

Review: When a casting is performed, lowest byte is copied first then the higher bytes.

```
#include <stdio.h>
#include <conio.h>
main()
{  int n=260, *p=&n;
   printf("n=%d\n",n);
   char *pp=(char*)p;
   *pp=0;
   printf("n=%d\n",n);
   getch();
}
```

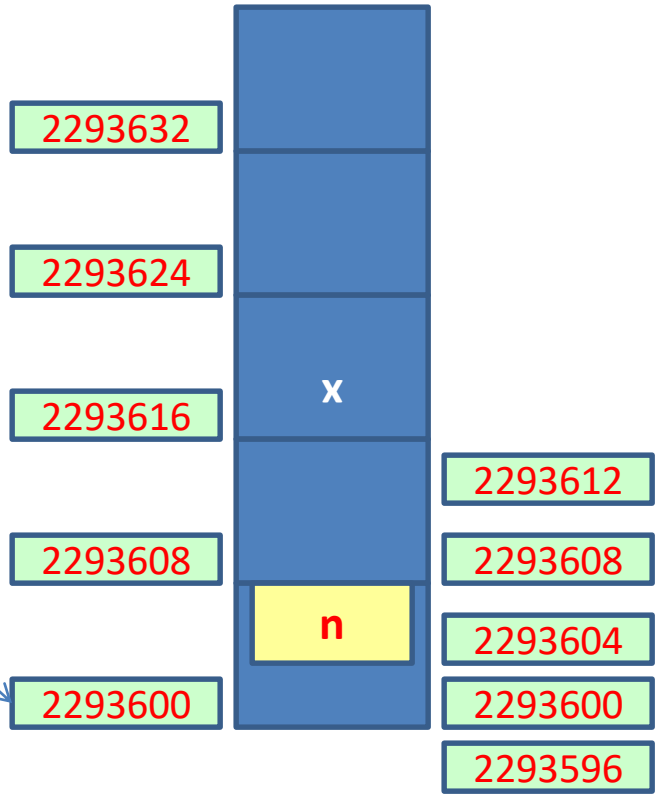
```
n=260
n=256
```



Pointer Arithmetic Operators : +, -, ++, --

```
1  #include <stdio.h>
2  int main()
3  {   double x = 0.5;
4      double* pD = &x;
5      int i;
6      for (i= -2; i<=2; i++) printf("%u, ", pD+i);
7      printf("\n");
8      int n= 3;
9      int *pI = &n;
10     for (i= -2; i<=2; i++) printf("%u, ", pI+i);
11     printf("\n");
12     pI++;
13     printf("%u\n", pI);
14     pI--;
15     printf("%u\n ", pI);
16     getchar();
17     return 0;
18 }
```

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



If access data using pD (bytes) can cause a harm on the variable n

```
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);
   getchar();
   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 Arithmetic Operators…

++ptr_var or ptr_var++	points to next **integer** after var
—ptr_var or ptr_var—	points to **integer** previous to var
ptr_var + i	points to the ith integer after var
ptr_var - i	points to the ith integer before var
++*ptr_var or (*ptr_var)++	will increment **var** by 1
*ptr_var++	will fetch the value of the next integer after var

- Each time a pointer is incremented, it points to the memory location of the next element of its base type
- Each time it is decremented it points to the location of the previous element
- All other pointers will increase or decrease depending on the length of the data type they are pointing to

Pointer Comparisons

- Two pointers can be compared in a relational expression provided both the pointers are pointing to variables of the same type.
- Consider that ptr_a and ptr_b are 2 pointer variables, which point to data elements a and b. In this case the following comparisons are possible:

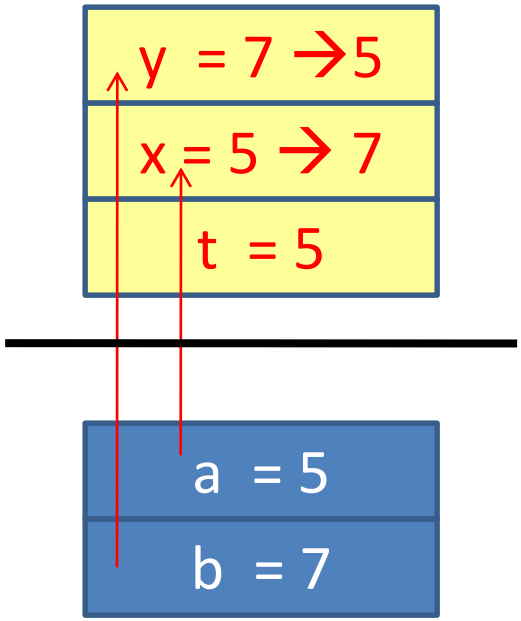
ptr_a < ptr_b	Returns true provided **a** is stored before **b**
ptr_a > ptr_b	Returns true provided **a** is stored after **b**
ptr_a <= ptr_b	Returns true provided **a** is stored before **b** or ptr_a and ptr_b point to the same location
ptr_a >= ptr_b	Returns true provided **a** is stored after **b** or ptr_a and ptr_b point to the same location.
ptr_a == ptr_b	Returns true provided both pointers ptr_a and ptr_b points to the same data element.
ptr_a != ptr_b	Returns true provided both pointers ptr_a and ptr_b point to different data elements but of the same type.
ptr_a == NULL	Returns true if ptr_a is assigned NULL value (zero)

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
```

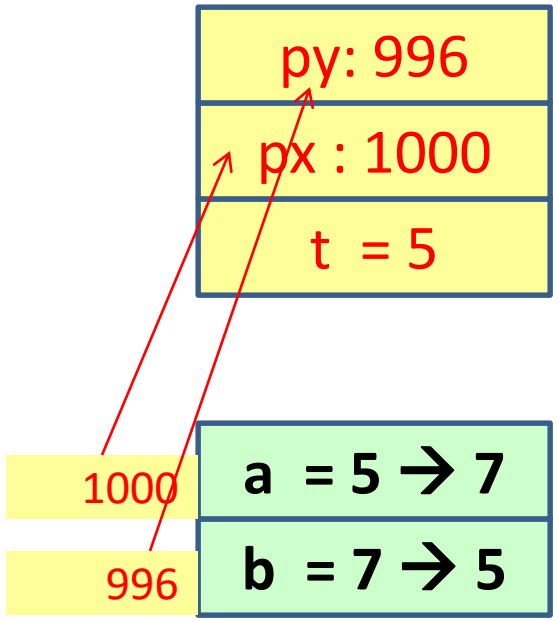


Pointer as Parameters of a Function …

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
```



8- Dynamic Allocated Data

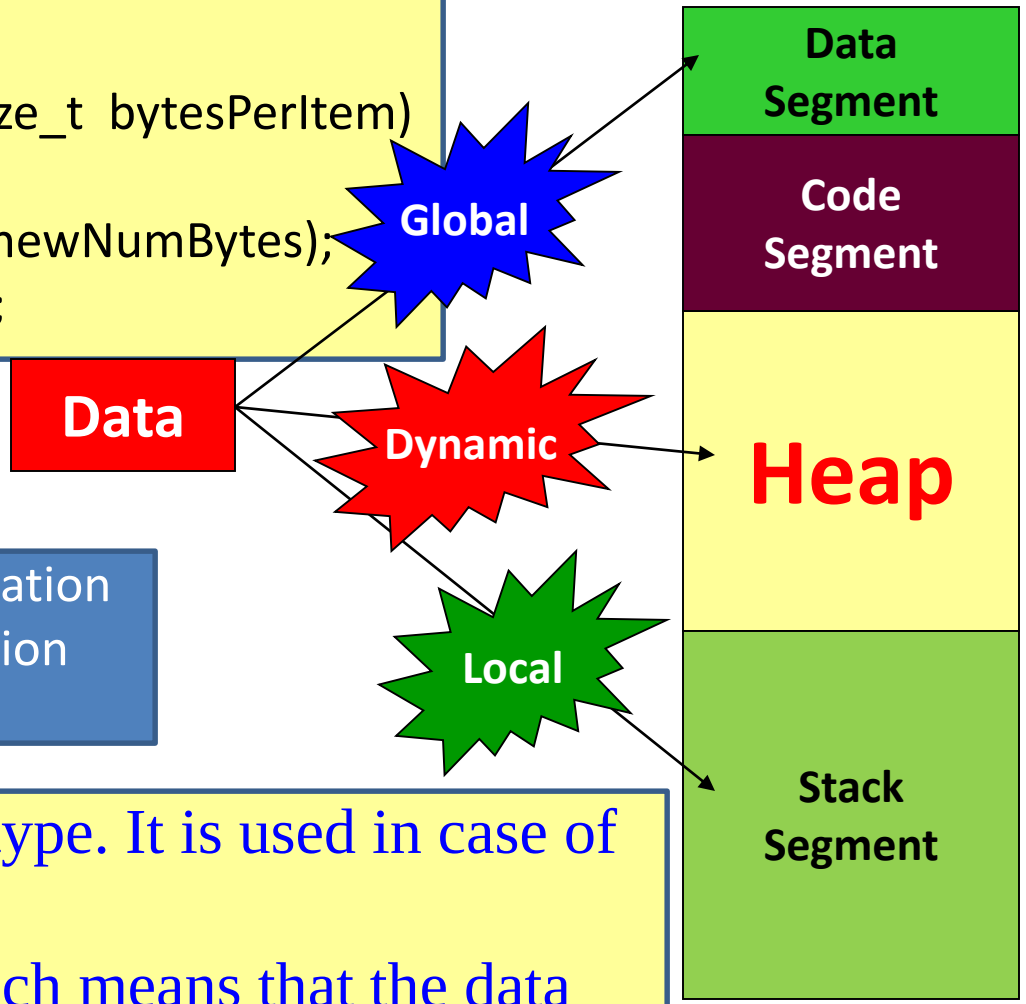
Stdlib.h

```
void*  calloc  (size_t numberOfItem, size_t  bytesPerItem)
void*  malloc (size_t  numBytes) ;
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
reall oc: 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>
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);
   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);
   getchar();
   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.

Example:

Input: DA

Output:

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

Danh từ: 2 pointer char* pc1, *pc2;
Động từ
-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.

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

Extra Walkthroughs with functions

•Study the following C-function:

```
int t (int x, int y, int z)
{   int k= 2*x + 3*y + 5*z;
    return k%13;
}
```

Suppose the above function is used in the following code:

```
int a=7, b=6, c=5;
int L= t(b,a,c);
```

What is the value of the L variable after this code is executed?

Extra Walkthroughs with functions

•Study the following C-function:

```
void T (int * p, int*q)
{   int t= *p; *p=*q; *q=t;
}
```

Suppose the above function is used in the following code:

```
int a=7, b=6;
T(&a,&b);
```

What are the values of the a and b variables after this code is executed?

Extra Walkthrough with functions

•Study the following C-function:

```
int T (int * p, int*q)
{   int t= (*p) + (*q) > 12 ? 5:6;
    return 2*t%5;
}
```

Suppose the above function is used in the following code:

```
int a=3, b=4, c;
c= T(&a,&b);
```

What is the value of the **C** variable after this code is executed?