

# Программирование на языке C++

...

Указатели и ссылки. Работа с памятью

# Ячейки памяти

Байт - минимальный адресуемый  
элемент запоминающего  
устройства ЭВМ

0	1	0	0	1	1	0	0	0	1	0	0	0	1	1	1
0	0	1	0	0	0	1	1	0	0	1	0	0	0	0	1
0	0	0	1	1	0	0	0	1	1	0	1	0	1	0	0
0	0	1	0	1	0	1	0	0	0	0	1	1	1	1	0

В современных машинах минимальным адресуемым блоком информации называется байт. Обычно байт состоит из 8 бит.

# Адрес ячейки памяти

Ячейки памяти имеют адрес (порядковый номер, число) по которому к ним могут обращаться команды процессора

Адрес	Значение
0x0000	1A
0x0001	90
0x0002	B7
0x0003	CC
....	
0xFFFFB	31
0xFFFFC	1D
0xFFFFD	AF
0xFFFFF	2C

# Адрес ячейки памяти

Адрес, как и содержимое ячейки памяти обычно записывается в шестнадцатеричном виде, но это тоже самое число

Адрес		Значение	
HEX	DEC	HEX	DEC
0x0000	0	1A	26
0x0001	1	90	144
0x0002	2	B7	183
0x0003	3	CC	204
....	....		
0xFFFFB	65532	31	49
0xFFFFC	65533	1D	29
0xFFFFD	65534	AF	175
0xFFFFF	65535	2C	44

# Диапазон адресов

Размер адреса	Мах адрес	
Байт (бит)	HEX	DEC (байт)
1 (8)	0xFF	256
2 (16)	0xFFFF	65'535
4 (32)	0xFFFFFFFF	4'294'967'295
6 (48)	0xFFFFFFFFFFFF	281'474'976'710'655

# Адрес ячейки памяти

Ячейки памяти имеют адрес (порядковый номер, число) по которому к ним могут обращаться команды процессора

```
#include <iostream>
```

```
int main()  
{
```

```
    short value{ 42 };
```

```
    std::cout << "value = " << value << std::endl;
```

```
    std::cout << "value address = " << &value << std::endl;
```

```
}
```

```
value = 42
```

```
value address = 0x74cc3a699c2c
```

```
#include <iostream>
```

```
int main()
```

```
{
```

```
    short value{ 42 };
```

```
    std::cout << "value = " << value << std::endl;
```

```
    std::cout << "value size = " << sizeof(value) << std::endl;
```

```
    std::cout << "value address = " << &value << std::endl;
```

```
}
```

```
value = 42
```

```
value size = 2
```

```
value address = 0x74cc3a699c2c
```

Адрес

Значение

0x74cc3a699c2a

1A

0x74cc3a699c2b

90

0x74cc3a699c2c

2A

0x74cc3a699c2d

00

0x74cc3a699c2f

1A

0x74cc3a699c30

90

0x74cc3a699c31

B7

0x74cc3a699c32

CC

# Указатель

Переменная, значением которой  
является адрес ячейки памяти

```
#include <iostream>
```

```
int main()
```

```
{
```

```
    short value{ 42 };
```

```
    short* ptrValue = &value;
```

```
    std::cout << "value = " << value << std::endl;
```

```
    std::cout << "value address = " << ptrValue << std::endl;
```

```
}
```

```
value = 42
```

```
value address = 0x74cc3a699c2c
```



```
#include <iostream>
```

```
int main()  
{
```

```
    short value{ 42 };
```

```
    short* ptrValue = &value;
```

```
    std::cout << "value = " << value << std::endl;
```

```
    std::cout << "value size = " << sizeof(value) << std::endl;
```

```
    std::cout << "value address = " << ptrValue << std::endl;
```

```
}
```

```
value = 42  
value size = 2  
value address = 0x74cc3a699c2c
```

**short\* ptrValue**



**Адрес**

**Значение**

0x74cc3a699c2a

**1A**

0x74cc3a699c2b

**90**

0x74cc3a699c2c

**2A**

0x74cc3a699c2d

**00**

0x74cc3a699c2f

**1A**

0x74cc3a699c30

**90**

0x74cc3a699c31

**B7**

0x74cc3a699c32

**CC**

```
#include <iostream>
```

```
int main()  
{
```

```
    int value{ 42 };
```

```
    int* ptrValue = &value;
```

```
    std::cout << "value = " << value << std::endl;
```

```
    std::cout << "value size = " << sizeof(value) << std::endl;
```

```
    std::cout << "value address = " << ptrValue << std::endl;
```

```
}
```

```
value = 42
```

```
value size = 4
```

```
value address = 0x74cc3a699c2c
```

**int\* ptrValue**



**Адрес**

**Значение**

0x74cc3a699c2a

**1A**

0x74cc3a699c2b

**90**

0x74cc3a699c2c

**2A**

0x74cc3a699c2d

**00**

0x74cc3a699c2f

**00**

0x74cc3a699c30

**00**

0x74cc3a699c31

**B7**

0x74cc3a699c32

**CC**

# Разыменование указателя

Операция получения доступа к  
значению переменной, на которую  
ссылается указатель

```
#include <iostream>
```

```
int main()
```

```
{
```

```
    short value{ 42 };
```

```
    short* ptrValue = &value;
```

```
    *ptrValue = 108;
```

```
    std::cout << "value address = " << ptrValue << std::endl;
```

```
    std::cout << "value = " << *ptrValue << std::endl;
```

```
}
```

```
value address = 0x74cc3a699c2c  
value = 108
```

```
int main()
{
    int value{ 0 };
    int* ptrValue = &value;

    *ptrValue = 720408;

    std::cout << value;
}
```

Адрес	Значение	Значение
0x74cc3a699c2a	1A	1A
0x74cc3a699c2b	90	90
0x74cc3a699c2c	00	18
0x74cc3a699c2d	00	FE
0x74cc3a699c2f	00	0A
0x74cc3a699c30	00	00
0x74cc3a699c31	B7	B7
0x74cc3a699c32	CC	CC

# Размер указателя

Не зависит от размера данных, на  
которые он указывает

```
int main()
{
    bool bVal {false};
    short sVal { 42 };
    double dVal { 3.1415 };

    std::cout << "bVal size = " << sizeof(bVal) << std::endl;
    std::cout << "sVal size = " << sizeof(sVal) << std::endl;
    std::cout << "dVal size = " << sizeof(dVal) << std::endl;

    std::cout << "bVal pointer size = " << sizeof(&bVal) << std::endl;
    std::cout << "sVal pointer size = " << sizeof(&sVal) << std::endl;
    std::cout << "dVal pointer size = " << sizeof(&dVal) << std::endl;
}
```

```
bVal size = 1
sVal size = 2
dVal size = 8
bVal pointer size = 8
sVal pointer size = 8
dVal pointer size = 8
```

# Размещение переменных в памяти

```
int main()
{
    bool bVal {false};
    short sVal { 42 };
    double dVal { 3.1415 };

    std::cout << "bVal pointer = " << &bVal << std::endl;
    std::cout << "sVal pointer = " << &sVal << std::endl;
    std::cout << "dVal pointer = " << &dVal << std::endl;
}
```

```
bVal pointer = 0x7c588628b9a5
sVal pointer = 0x7c588628b9a6
dVal pointer = 0x7c588628b9a8
```

```

int main()
{
    bool bVal {false};
    short sVal { 42 };
    double dVal { 3.1415 };

    std::cout << "bVal size = " << sizeof(bVal) << "\n";
    std::cout << "sVal size = " << sizeof(sVal) << "\n";
    std::cout << "dVal size = " << sizeof(dVal) << "\n";

    std::cout << "bVal pointer = " << &bVal << "\n";
    std::cout << "sVal pointer = " << &sVal << "\n";
    std::cout << "dVal pointer = " << &dVal << "\n";
}

```

```

bVal size = 1
sVal size = 2
dVal size = 8
bVal pointer = 0x7c588628b9a5
sVal pointer = 0x7c588628b9a6
dVal pointer = 0x7c588628b9a8

```

	Адрес	Значение
&bVal	0x7c588628b9a5	00
	0x7c588628b9a6	2A
&dVal	0x7c588628b9a7	00
	0x7c588628b9a8	10
	0x7c588628b9a9	4D
	0x7c588628b9aa	00
	0x7c588628b9ab	B7
	0x7c588628b9ac	CC
	0x7c588628b9ad	00
	0x7c588628b9ae	B7
	0x7c588628b9af	CC

# Проверка типа

Производится на этапе компиляции  
программы

```
int main()
{
    int value{123};
    int* pValue = &value;

    pValue = 1024;
}
```

In function 'int main()':  
error: invalid conversion from 'int' to 'int\*' [-fpermissive]



# Адресная арифметика

```
int main()
{
    int value = 2048;
    int *ptr = &value;

    std::cout << ptr << '\n';
    std::cout << ptr+1 << '\n';
    std::cout << ptr+2 << '\n';
}
```

```
0x7f52abad1ddc
0x7f52abad1de0
0x7f52abad1de4
```

```

int main()
{
    int value = 1234567;
    int *ptr = &value;

    std::cout << ptr << '\n';
    std::cout << ptr+1 << '\n';
    std::cout << ptr+2 << '\n';

    std::cout << *ptr << '\n';
    std::cout << *(ptr + 1) << '\n';
}

```

```

0x7f52abad1ddc
0x7f52abad1de0
0x7f52abad1de4
1234567
-1803175099

```

sizeof(int)

sizeof(int)

sizeof(int)

Адрес

Значение

0x7f52abad1ddc

87

0x7f52abad1ddd

D6

0x7f52abad1dde

12

0x7f52abad1ddf

00

0x7f52abad1de0

??

0x7f52abad1de1

??

0x7f52abad1de2

??

0x7f52abad1de3

??

0x7f52abad1de4

??

0x7f52abad1de5

??

0x7f52abad1de6

??

0x7f52abad1de7

??

```

int main()
{
    bool value = true;
    bool *ptr = &value;

    std::cout << ptr << '\n';
    std::cout << ptr+1 << '\n';
    std::cout << ptr+2 << '\n';
}

```

```

0x7f52abad1ddc
0x7f52abad1ded
0x7f52abad1dee

```

	Адрес	Значение
sizeof(bool) ↑↓	0x7f52abad1ddc	01
sizeof(bool) ↑↓	0x7f52abad1ddd	??
sizeof(bool) ↑↓	0x7f52abad1dde	??

# Работа с указателями

```
int main()
{
    short value = 25;
    short *ptr = &value;
    short *ptr2 = &value;
    short *ptr3 = ptr2;

    *ptr2 = 100;

    std::cout << value << '\n';
    std::cout << *ptr << '\n';
    std::cout << *ptr2 << '\n';
    std::cout << *ptr3 << '\n';

    ptr3 = nullptr;
}
```

short value = 25;

value:

25

short \*ptr = &value;

ptr

short \*ptr2 = &value;

ptr2

short \*ptr3 = ptr2;

ptr3

25

\*ptr2 = 100;

ptr

100

ptr2

ptr3

ptr3 = nullptr;

ptr

100

ptr2

ptr3

nullptr

# Массивы

```
int main()
{
    int array[] = { 4, 9, 16, 25 };
    int arraySize = sizeof(array) / sizeof(array[0]);

    std::cout << "arraySize is " << arraySize << '\n';
    int* itemPtr = &array[0];
    while(itemPtr != &array[arraySize])
    {
        std::cout << "Val: " << *itemPtr << " Adr: " << itemPtr << '\n';
        ++itemPtr;
    }
}
```

```
arraySize is 4
Val: 4 Adr: 0x73d09ea40940
Val: 9 Adr: 0x73d09ea40944
Val: 16 Adr: 0x73d09ea40948
Val: 25 Adr: 0x73d09ea4095c
```

0x73d09ea40934	??	??	??	??
0x73d09ea40938	??	??	??	??
0x73d09ea4093c	??	??	??	??
&array[0] = 0x73d09ea40940	04	00	00	00
&array[1] = 0x73d09ea40944	09	00	00	00
&array[2] = 0x73d09ea40948	10	00	00	00
&array[3] = 0x73d09ea4094c	19	00	00	00
0x73d09ea40950	??	??	??	??
0x73d09ea40954	??	??	??	??
0x73d09ea40958	??	??	??	??
0x73d09ea4096c	??	??	??	??

## Передача указателей в функции

```
int* GetMinItem(int* beginPtr, int* endPtr)
```

```
int main()
```

```
{
```

```
    const size_t N = 8;
```

```
    int array[N] = { 10, -20, 30, 40, 50, -60, 0, -30 };
```

```
    int* minPtr = GetMinItem(array, array + N);
```

```
    std::cout << "Min value is " << *minPtr << "\n";
```

```
    std::cout << "Position in array is " << minPtr - array << "\n";
```

```
    return 0;
```

```
}
```



```
const size_t N = 8;
```

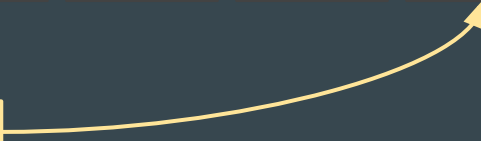
```
int array[N] = { 10, -20, 30, 40, 50, -60, 0, -30 };
```



```
int* minPtr = GetMinItem( array, array + N );
```



```
int* minPtr
```



```
int* GetMinItem(int* beginPtr, int* endPtr)
{
    if(beginPtr > endPtr) {
        return nullptr;
    }

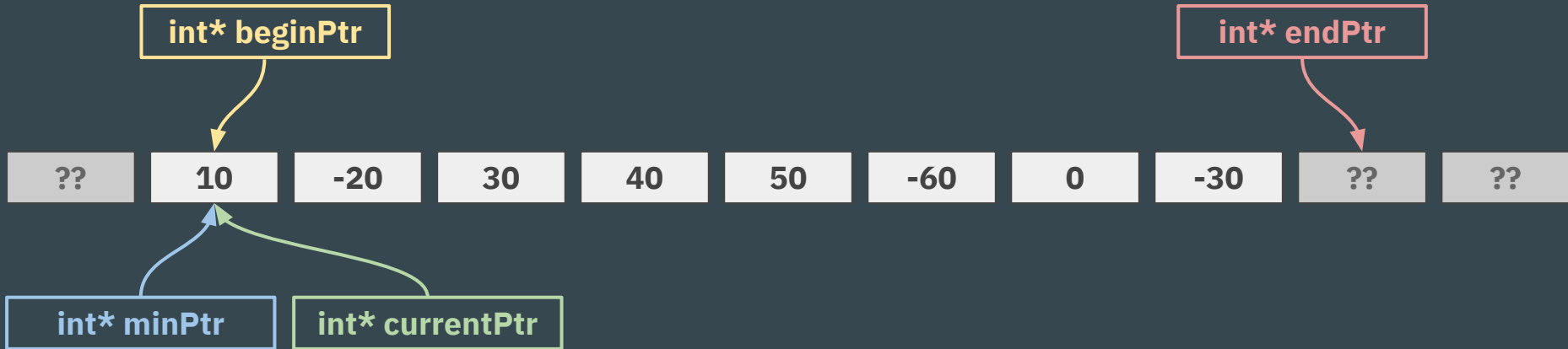
    int* currentPtr = beginPtr;
    int* minPtr = beginPtr;

    while(currentPtr++ != endPtr)
    {
        if(*currentPtr < *minPtr)
        {
            minPtr = currentPtr;
        }
    }

    return minPtr;
}
```

```
int* GetMinItem(int* beginPtr, int* endPtr)
{
    if(beginPtr > endPtr) {
        return nullptr;
    }
}
```

```
int* currentPtr = beginPtr;
int* minPtr = beginPtr;
```





```
while(currentPtr++ != endPtr)
{
    if(*currentPtr < *minPtr)
    {
        minPtr = currentPtr;
    }
}
```



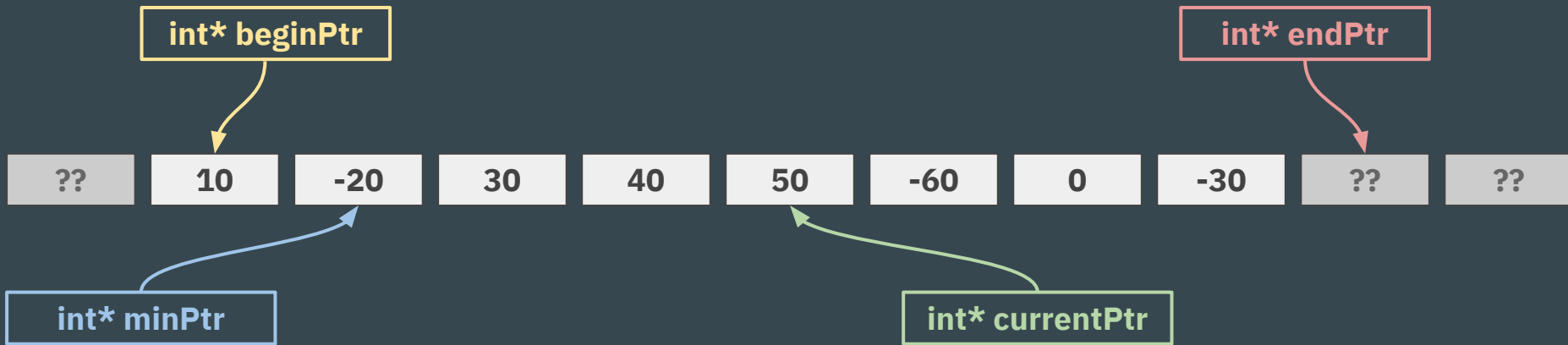
```
while(currentPtr++ != endPtr)
{
    if(*currentPtr < *minPtr)
    {
        minPtr = currentPtr;
    }
}
```



```
while(currentPtr++ != endPtr)
{
    if(*currentPtr < *minPtr)
    {
        minPtr = currentPtr;
    }
}
```

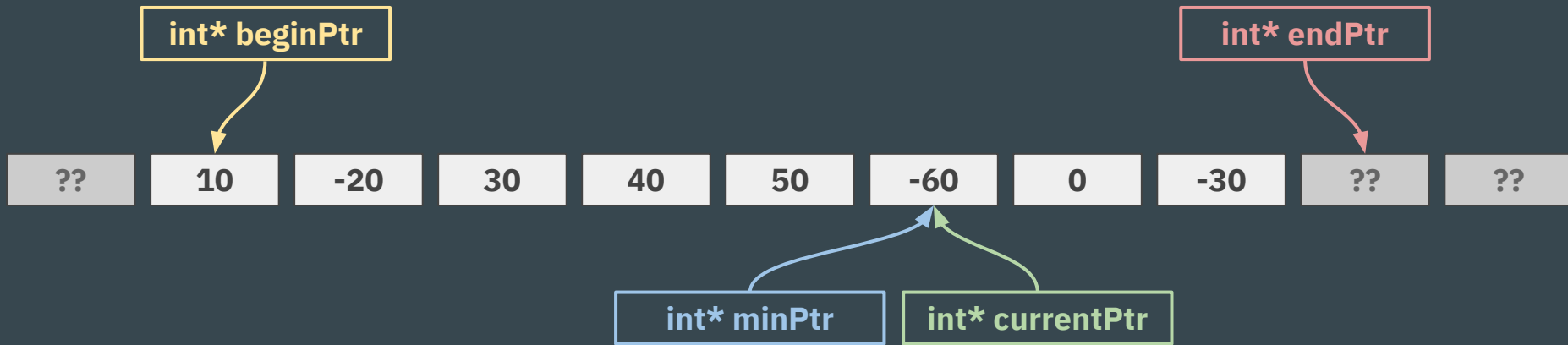


```
while(currentPtr++ != endPtr)
{
    if(*currentPtr < *minPtr)
    {
        minPtr = currentPtr;
    }
}
```

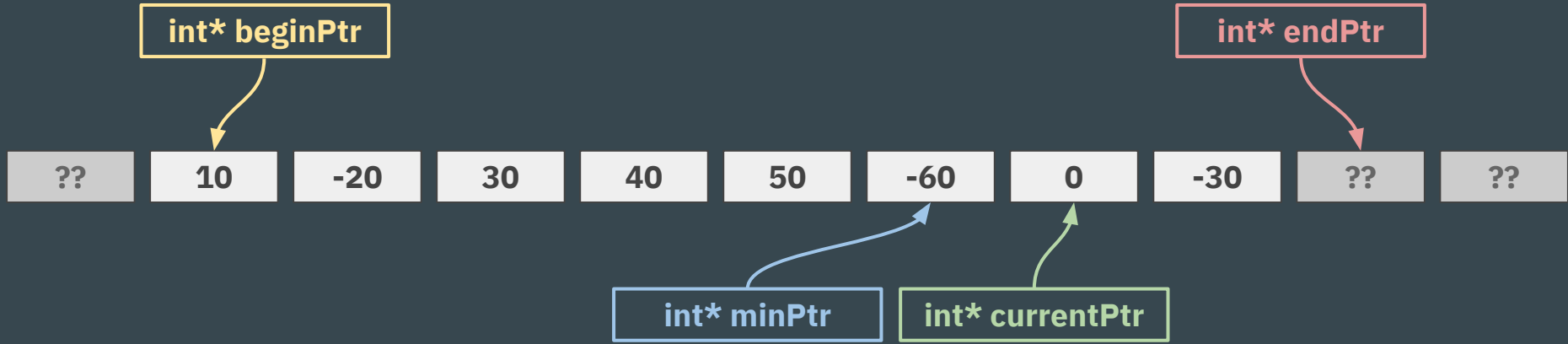


```
while(currentPtr++ != endPtr)
{
    if(*currentPtr < *minPtr)
    {
        minPtr = currentPtr;
    }
}
```

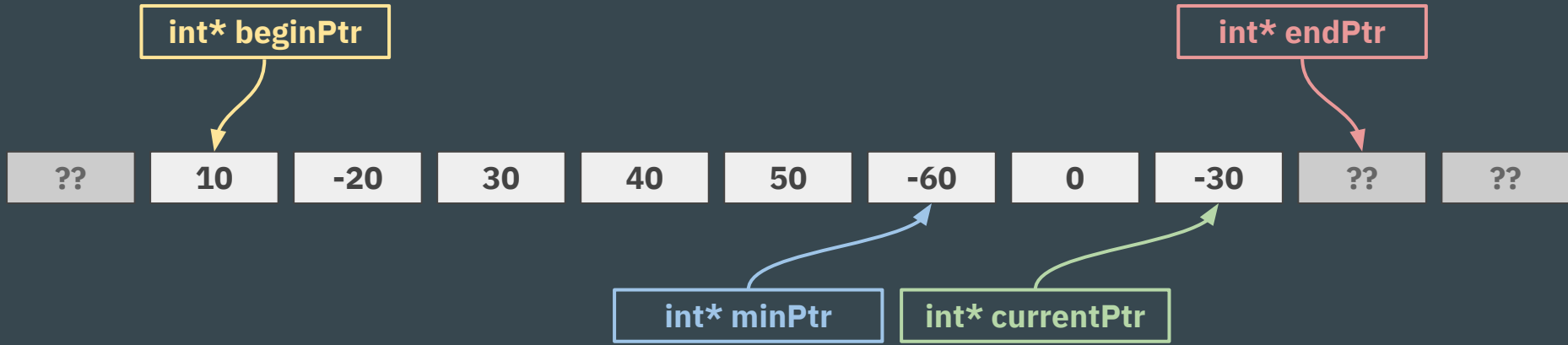




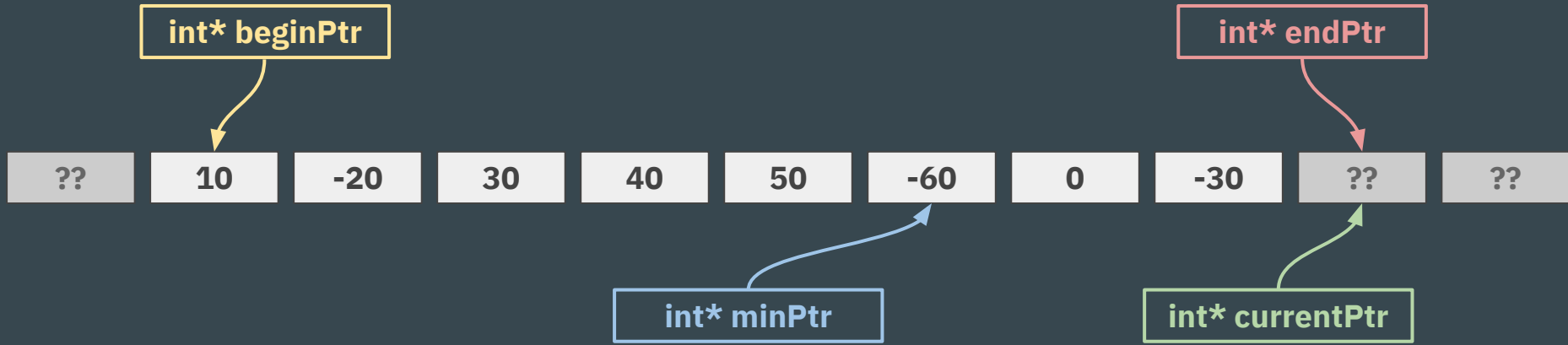
```
while(currentPtr++ != endPtr)
{
    if(*currentPtr < *minPtr)
    {
        minPtr = currentPtr;
    }
}
```



```
while(currentPtr++ != endPtr)
{
    if(*currentPtr < *minPtr)
    {
        minPtr = currentPtr;
    }
}
```



```
while(currentPtr++ != endPtr)
{
    if(*currentPtr < *minPtr)
    {
        minPtr = currentPtr;
    }
}
```



```
while(currentPtr++ != endPtr)
{
    if(*currentPtr < *minPtr)
    {
        minPtr = currentPtr;
    }
}
```