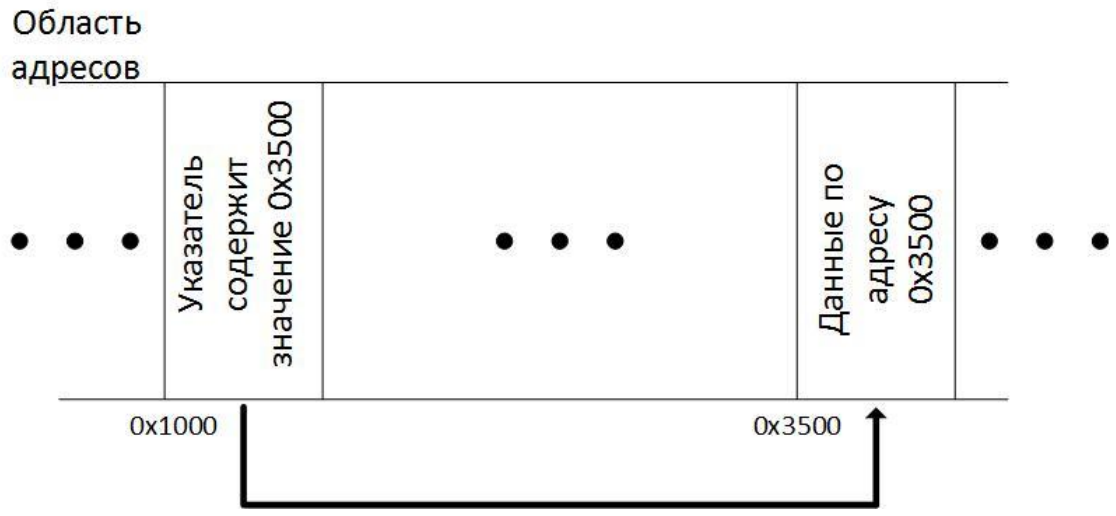
Лекция 1.
Указатели

Понятие указателя

Указатель (pointer) – переменная, которая хранит в себе адрес области в памяти или специального значения – нулевого адреса (NULL).



Объявление указателя

<тип указателя> *<имя указателя>

Указатели объявляются с помощью символа "*", который добавляется после типа указателя.

Примеры:

1) int* number_ptr;

2) double* double_ptr;

Указатели, как и переменные, если их не инициализировать будут содержать случайное значение. И такие указатели могут обратиться к недопустимой памяти, что может привести к отказу приложения или некорректной модификации данных. Поэтому, лучше при объявлении указателя инициализировать его нулевым адресом – NULL.

Примеры:

1) int* number_ptr = NULL;

2) double* double_ptr = NULL;

Инициализация указателя

Также можно инициализировать указатель адресом уже существующей переменной.

```
int Age = 18; // Инициализация переменной
int *pAge = &Age; // Инициализация указателя

double length = 7.77; // Инициализация переменной
double *pLength = &length; // Инициализация указателя

std::cout << "Value: " << Age << " Address: " << &Age << "\n";
std::cout << "Value: " << *pAge << " Address: " << pAge << "\n";

std::cout << "Value: " << length << " Address: " << &length << "\n";
std::cout << "Value: " << *pLength << " Address: " << pLength << "\n";
```

Результат

Value: 18 Address: 0x755f7830c954
Value: 18 Address: 0x755f7830c954
Value: 7.77 Address: 0x755f7830c958
Value: 7.77 Address: 0x755f7830c958

Переопределение указателя

```
int Age = 18; // Инициализация переменной
int *pAge = &Age; // Инициализация указателя

std::cout << "Value: " << *pAge << " Address: " << pAge << "\n";

int Age2 = 40;
pAge = &Age2; // Переопределение указателя

std::cout << "Value: " << *pAge << " Address: " << pAge << "\n";

*pAge += 10;

std::cout << "Value: " << *pAge << " Address: " << pAge << "\n";
std::cout << "Value: " << Age2 << " Address: " << &Age2 << "\n";

Age2 *= 2;

std::cout << "Value: " << *pAge << " Address: " << pAge << "\n";
std::cout << "Value: " << Age2 << " Address: " << &Age2 << "\n";
```

Результат

Value: 18 Address: 0x7774d6eadfe8
Value: 40 Address: 0x7774d6eadfec
Value: 50 Address: 0x7774d6eadfec
Value: 50 Address: 0x7774d6eadfec
Value: 100 Address: 0x7774d6eadfec
Value: 100 Address: 0x7774d6eadfec

(*) – разыменование(оператор обращения к значению) указателя. То есть получение значения по адресу который содержится в указателе.

(&) – оператор обращения к адресу.

Использование ключевого слова const с указателями

- Данные, на которые указывает указатель, являются постоянными и не могут быть изменены, но сам адрес, содержащийся в указателе, вполне может быть изменен (т.е. указатель может указывать и на другое место):

```
int HoursInDay = 24;
const int* pInteger = &HoursInDay; // нельзя использовать pInteger
// для изменения HoursInDay
int MonthsInYear = 12;
pInteger = &MonthsInYear; // ok
*pInteger =13; // Ошибка компиляции: нельзя изменять данные
int* pAnotherPointerToInt = pInteger; // Ошибка компиляции: нельзя присвоить константу не константе
```

-Содержащийся в указателе адрес является постоянным и не может быть изменен, однако данные, на которые он указывает, вполне могут быть изменены:

```
int DaysInMonth = 30;
// pInteger не может указать ни на что иное
int* const pDaysInMonth = &DaysInMonth;
*pDaysInMonth = 31; // OK! Значение может быть изменено
int DaysInLunarMonth = 28;
pDaysInMonth = &DaysInLunarMonth; // Ошибка компиляции: нельзя изменить адрес!
```

- И содержащийся в указателе адрес, и значение, на которое он указывает, являются константами и не могут быть изменены (самый ограничивающий вариант):

```
int HoursInDay = 24;
// указатель может указать только на HoursInDay
const int* const pHoursInDay = &HoursInDay;
*pHoursInDay = 25; // Ошибка компиляции: нельзя изменить значение, на которое указывает указатель
int DaysInMonth = 30;
pHoursInDay = &DaysInMonth; // Ошибка компиляции: нельзя изменить значение указателя
```

Адресная арифметика

Над указателями возможны такие операции:

- ✔ присвоения;
- ✔ сравнения;
- ✔ увеличение/уменьшение;

Пример увеличения указателя:

```
int arr[ 5 ] = { 1, 5, 10, 15, 20 };
int *p = &arr[0];
int *p2 = &arr[0] + 2;

std::cout << "Address arr[0]: " << p << " Value: " << *p << "\n";
std::cout << "Address arr[0] + 2: " << p2 << " Value: " << *p2 << "\n";
```

Результат

Address arr[0]: 0x725d8f0c27f0 Value: 1
Address arr[0] + 2: 0x725d8f0c27f8 Value: 10

Пример использования указателя с массивами:

```
const int arr_size = 5;
int arr[ arr_size ] = { 1, 5, 10, 15, 20 };

std::cout << "Address arr[0]: " << &arr[0] << " Address arr: " << arr << "\n";

for (int i = 0; i < arr_size; i++)
{
  std::cout << "Value arr[i]: " << arr[i] << " Value arr: " << *(arr + i) << "\n";
}
```

Результат

```
Address arr[0]: 0x709e9d0b4d80 Address arr: 0x709e9d0b4d80
Value arr[i]: 1 Value *(arr + i): 1
Value arr[i]: 5 Value *(arr + i): 5
Value arr[i]: 10 Value *(arr + i): 10
Value arr[i]: 15 Value *(arr + i): 15
Value arr[i]: 20 Value *(arr + i): 20
```

Операторы new и delete

Оператор new используется для резервирования (распределения) новых блоков памяти. Чаще всего используется версия оператора new, возвращающая указатель на затребованную область память в случае успеха, и передающая исключение в противном случае.

Тип* *Указатель* **= new** *Тип*; *//* запрос памяти для одного элемента
Тип* *Указатель* **= new** *Тип [КолЭлементов]*; *//* запрос памяти для нескольких элементов

Таким образом, если необходимо разместить в памяти целые числа, используйте следующий код:

```
int* pNumber = new int; // получить указатель на целое число
int* pNumbers = new int[10];  // получить указатель на блок из 10 целых чисел
```

Операторы new и delete

Каждая область памяти, зарезервированная оператором new, должна быть в конечном счете освобождена (очищена) соответствующим оператором delete:

Тип* *Указатель* **= new Тип;**
delete *Указатель*; *//* освобождение памяти, зарезервированной ранее для одного экземпляра Типа

Это правило применимо также при запросе памяти для нескольких элементов:
Тип* *Указатель* **= new** *Тип [КолЭлементов]*;
delete[] *Указатель*; *//* освободить зарезервированный ранее блок