Тема 3-2.

Указатели и ссылки

для АСУб и ЭВМб

Темы лекции

- Типы памяти: статическая, автоматическая, динамическая
- Указатели и ссылки
- Умные указатели в C++
- Динамические массивы
- Сравнение динамических и статических массивов

Повторение: объявление переменной

- **Что означает запись: int A=10;**
- **Доступ к объявленной переменной осуществляется *по ее имени*.**
- **При этом все обращения к переменной заменяются на адрес ячейки памяти, в которой хранится ее значение.**
- **При завершении программы или функции, в которой была описана переменная, память автоматически освобождается.**

Повторение: память

- В современных ЭВМ наименьшей адресуемой структурной единицей информации принят байт и байтовая организация информации в оперативной памяти (ОП).
- Для представления алфавитно-цифровой информации в ЭВМ обычно используется **машинное слово** – совокупность символов, которая считывается из ОП или записывается в нее за одно обращение. Обычно машинное слово содержит целое число байтов. Байты адресуются последовательно, начиная с нуля.

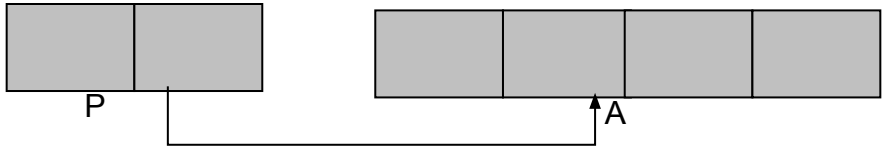
Повторение: адресация

- Для многобайтовых *шин данных* (т.е. начиная с 16-битной) физическая адресация памяти может происходить по словам, т.е. на *шину адреса* всегда подаётся адрес слова, а *шина данных* считывает или записывает нужную его часть — от отдельного байта до слова целиком.
- Для обозначения разрядности доступа может применяться отдельная шина байт-маски (по биту на каждый байт шины данных)

Указатели

- Указатель – это переменная, в которой хранится адрес другой переменной или участка памяти.
- Объявление указателей:
  - Как и любая переменная, *указатель должен быть объявлен*.
  - При описании переменных-указателей перед именем переменной ставится «*».
  - При объявлении указателей всегда указывается тип объекта, который будет храниться по данному адресу:
  - **тип *имя_переменной;**
  - **Пример: int *a;**
- Звездочка в описании указателя относится непосредственно к имени, поэтому, чтобы объявить несколько указателей, ее ставят перед именем каждого из них:
  - **float *x, y, *z;**

- с помощью операции получения адреса
  - int a=5;
  - int* p=&a; **// или int p(&a);**



- с помощью проинициализированного указателя
  int* r=p;
- адрес присваивается в явном виде
  char* cp=(char*)0x B800 0000;
  где 0x B800 0000 – шестнадцатеричная константа,
  (char*) – операция приведения типа.
- присваивание пустого значения:
  int* N=NULL;
  int* R=0;

Операция получения адреса &

- Операция получения адреса обозначается знаком &.
- Возвращает адрес своего операнда.
  - **float a; //объявлена вещественная переменная a**
  - **float *adr_a; //объявлен указатель на тип float**
  - **adr_a = &a; //оператор записывает в переменную *adr_a* адрес переменной *a***

Операция разадресации (разыменования) *

- *Операция разадресации* * возвращает значение переменной, хранящееся по заданному адресу, т.е. выполняет действие, обратное операции &:
  - float a; //Объявлена вещественная переменная *a*
  - float *adr_a; //Объявлен указатель на тип float
  - a=*adr_a; //Оператор записывает в переменную *a* вещественное значение, хранящееся по адресу *adr_a*.

Операции * и & при работе с указателями

Описание	Адрес	Значение, хранящееся по адресу
тип *p;	p	*p
тип p;	&p	p

Понятие об указателях

```
char C = '$'; // будет выделена память под переменную C
              // и ей присвоено начальное значение
 cout << C; // из ячейки памяти с именем C будет извлечено
            // значение и выведено на экран
```

- **Синтаксис объявления указателя:**

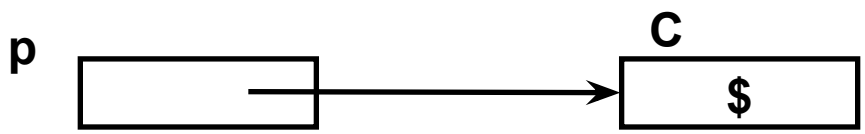
```
тип_данных *имя_переменной;
```

- **Пример:**

```
float *x, *y, *z;
char *p, ch;
```

- **Пример использования операции получения адреса (&) и операции разыменования (*)**

```
p = &C; //в указатель p записывается адрес переменной C
ch = *p; // в переменную ch записывается символьное
         // значение, хранящееся по адресу p
```



Понятие об указателях

- Как правило, при обработке оператора описания переменной компилятор автоматически выделяет память под переменную в соответствии с указанным типом. При завершении программы или функции, в которой была описана переменная, память автоматически освобождается.
- Доступ к объявленной переменной осуществляется по ее имени. При этом все обращения к переменной меняются на адрес ячейки памяти, в которой хранится ее значение.

- Доступ к значению переменной можно получить иным способом – определить **собственные переменные** для хранения адресов памяти. Такие переменные называют *указателями*.
- Итак, *указатель* – это переменная, значением которой является адрес памяти, по которому храниться объект определенного типа (другая переменная).

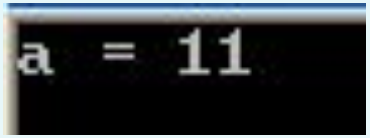
- Как и любая переменная, указатель должен быть объявлен. При объявлении указателей всегда указывается тип переменной, значение которой будет храниться по данному адресу. Звездочка в описании указателя относиться непосредственно к имени, поэтому, чтобы объявить несколько указателей, ее ставят перед именем каждого из них.

Понятие о ссылках

- **Формат описания ссылки:**

тип &идентификатор_1 = идентификатор_2;

```
#include <iostream>
using namespace std;
int main(void)
{ int a = 5, b = 10;
  int &aRef = a; // aRef является ссылкой на a
  aRef = b; // a равно b
  aRef++;// a++;
  cout << "a = " << a << endl;
}
```



Понятие о ссылках

- Ссылка на некоторую переменную может рассматриваться как указатель, который при работе с ним всегда разыменовывается. Для ссылки не требуется дополнительного пространства в памяти: она является просто другим именем или псевдонимом переменной. Для определения ссылки применяется унарный оператор &.

- Ссылка не создает копию объекта, а лишь является другим именем объекта. Чаще всего ссылки используются для передачи параметров в функции.

Ссылки

Ссылка (reference) – является альтернативным именем переменной, указанной при инициализации ссылки. Ссылка является переменной, которая содержит адрес другой переменной. По существу – это неявный указатель с константным значением адреса.

Особенности ссылок:

1. Ссылка при объявлении обязательно должна быть **проинициализирована**.
2. Значение ссылки **не** может быть **изменено** в ходе работы программы.
3. Для получения данных по ссылке **не** надо пользоваться операцией **разыменовывания**.
4. Нельзя создавать указатель на ссылку (**у ссылки нет адреса**).
5. Нельзя создавать массивы ссылок.

```
Тип& ИмяСсылки (ИмяЯвнойПеременной) ;
```

Ссылки

```
int x = 10;
int& rX = x;
int y = rX;

rX = 20;

std::cout << x << std::endl;//20
std::cout << y << std::endl;//10
std::cout << rX << std::endl;//20

const int& crX = x;
// crX ++;

// int& r = x + 10;
```

Определения

Указатель (pointer) – это переменная, значением которой является адрес другой переменной.
Тип указателя обязательно должен совпадать с типом переменной, адрес которой он хранит.

Применение указателей предоставляет возможность создавать динамические структуры данных, размер которых определяется не при компиляции программы, а в процессе ее исполнения.

Тип* Идентификатор ;
Тип *Идентификатор ;
Тип *Идентификатор1, …, *ИдентификаторN ;

```
char* ps ;
float *ptr ;
int *px, *py ;
char* p, ch ;
char* ps2 (0) ;
int *px2 = 0;
```

Нельзя использовать в программе указатель, значение которого не определено.

Размер указателей:

- ***Размер указателя*** зависит **от архитектуры**, на которой скомпилирован исполняемый файл.
- Следовательно, указатель на 32-битном устройстве занимает **32 бита** (4 байта). С 64-битным исполняемым файлом указатель будет занимать **64 бита** (8 байт).
- Это **вне зависимости** от того, **на что указывает** указатель.

Нулевое значение и нулевые указатели

- Помимо адресов памяти, есть еще одно значение, которое указатель может хранить: значение **nullptr** или **NULL** или **0**.

- Нулевое значение (или «**значение null**») — это **специальное значение**, которое означает, что указатель **ни на что не указывает**.

- Указатель, содержащий нулевое значение, называется **нулевым указателем**.

Нулевое значение и нулевые указатели

- В языке C++ мы можем **присвоить** указателю нулевое значение, **инициализируя его/присваивая** ему литерал **0**:

- `int *ptr(0);` // ptr теперь нулевой указатель

- `int *ptr1;` // ptr1 не инициализирован
- `ptr1 = 0;` // ptr1 теперь нулевой указатель

Нулевое значение и нулевые указатели

- Поскольку значением нулевого указателя является **нуль**, то это ***можно использовать внутри условного ветвления*** для проверки того, ***является ли указатель нулевым или нет***:

```
#include <iostream>

int main()
{
    double *ptr(0);
    if (ptr)
        std::cout << "ptr is pointing to a double value.";
    else
        std::cout << "ptr is a null pointer.";
    return 0;
}
```

void*

```
int a = 10;
int *pA = &a;
//float *pF = &a;

void *pV = &a;
//std::cout << *pV << std::endl;

std::cout << (*(int*)pV) << std::endl;

//int* pB = pV;
int* pC = (int*)pV;

*pC = 20;
```

Арифметика указателей

- int a = 1, b = 2, *aa = &a,*bb = &b;

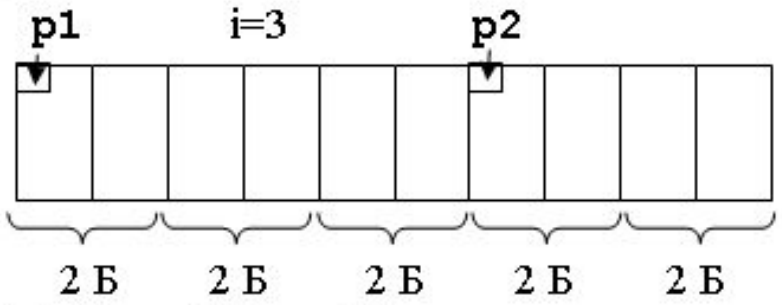
- cout << bb << endl;
- cout << aa << endl;
- //cout << aa << endl;

- //aa = nullptr;

- cout << (aa-bb)<<"\t"<<((long)aa-(long)bb)/sizeof(int);

Арифметические операции над указателями:

- *сложение* и *вычитание указателей с константой*;
- *вычитание одного указателя из другого*;
- *инкремент*; *декремент*.

ОПЕРАЦИЯ	ПРИМЕР	ОПИСАНИЕ
<указатель>+<целое>	`p2=p1+i;`	В результате этой операции значением переменной **p2** будет являться адрес области памяти, отстоящей от области памяти, адрес которой хранится в переменной **p1**, на количество единиц типа, на данные которого указывают эти указатели:  **i*sizeof**(*<тип>*)

ОПЕРАЦИЯ	ПРИМЕР	ОПИСАНИЕ
<указатель>–<целое>	`p2=p1-i;`	Аналогично сложению
<указатель>++; <указатель>– –;	`p1++;` `p2--`	Изменения происходят в единицах типа: `1·sizeof(`<тип>`);`
<указатель>– <указатель>	`i=p2-p1;`	При выполнении этой операции указатели должны быть одного и того же типа; результат – целое число, абсолютная величина которого указывает, сколько единиц данного типа помещается между адресом памяти первого и второго указателей. Знак этого числа показывает, какой из указателей больше.

Арифметика указателей

- int a = 1, b = 2, *aa = &a,*bb = &b;

- cout << bb << endl;
- cout << aa << endl;
- //cout << aa << endl;

- //aa = nullptr;

- cout << (aa-bb)<<"\t"<<((long)aa-(long)bb)/sizeof(int);

Сравнение указателей

Два указателя **могут быть сравнены** с помощью операций сравнения, если они указывают на переменные ***одного и того же типа***

Отношения	== != > < >= <=	Сравнить указатели. Возвращается истина, если операция сравнения закончилась успешно, иначе возвращается ложь. Сравниваемые указатели должны быть каким-то образом связаны, иначе операция не имеет смысла.

Операции с указателями

Название	Знак	Пояснения
Взятие адреса	**&**	Получить адрес переменной.
Разыменовывание	*	Получить значение переменной по её адресу.
Выделение памяти	**new**	Получить указатель на начало выделенного блока оперативной памяти.
Освобождение памяти	**delete**	Освободить выделенный блок памяти и сделать указатель недоступным.

Операции * и & при работе с указателями

Описание	Адрес	Значение, хранящееся по адресу
тип *p;	p	*p
тип p;	&p	p