УКАЗАТЕЛИ И ССЫЛКИ

УКАЗАТЕЛИ И ССЫЛКИ.

Программист может определить собственные переменные для хранения адресов областей памяти.

Такие переменные называются указателями.

Для удобства записи программного кода и улучшения читаемости программы в языке С определено понятие ссылки.

*Ссылка представляет собой **синоним имени**, указанного при инициализации ссылки*

Указатели

- Инициализация указателей.
- Операции с указателями.

УКАЗАТЕЛИ.

- *Указатель не является самостоятельным типом.*
- *Указатель всегда связан с каким-либо другим конкретным типом.*
- *Величины типа указатель подчиняются общим правилам определения области действия, видимости и времени жизни.*

- В C++ различают три вида указателей:
  - указатели на объект
  - указатели на функцию
  - указатели на **void**

Каждый вид указателей отличается своими свойствами и набором допустимых операций.

Указатель на функцию

содержит адрес в сегменте кода, по которому располагается исполняемый код функции, то есть адрес, по которому передается управление при вызове функции

- Указатели на функции используются для косвенного вызова функции, а также для передачи имени функции в другую функцию в качестве параметра

Синтаксис: **тип (*имя) (список_типов_аргументов);**

Например: **int (*fun) (double, double);**

задает указатель с именем **fun** на функцию, возвращающую значение типа **int** и имеющую два аргумента типа **double.**

Указатель на объект

содержит адрес области памяти, в которой хранятся данные определенного типа

- основного или составного

Простейшее объявление указателя на объект имеет вид:

тип *имя;

тип может быть любым, кроме ссылки и битового поля, причем тип может быть к этому моменту только объявлен, но еще не определен

- *следовательно, в **структуре** может присутствовать указатель на структуру того же типа*

Указатели

Звездочка относится непосредственно к имени, поэтому для того, чтобы объявить несколько указателей, требуется ставить ее перед именем каждого из них.

Например: **int *a, b, *c;**

описываются два указателя на целое с именами **a** и **c**, а также целая переменная **b**

- Размер указателя зависит от модели памяти.
- Можно определить указатель на указатель и т. д.

Указатель на **void**

применяется в тех случаях, когда конкретный тип объекта, адрес которого требуется хранить, не определен

- Например, если в одной и той же переменной в разные моменты времени требуется хранить адреса объектов различных типов

Указателю на **void** можно присвоить значение указателя любого типа, а также сравнивать его с любыми указателями.

Перед выполнением каких-либо действий с областью памяти, на которую ссылается указатель на **void**, требуется преобразовать его к конкретному типу явным образом

Указатель может быть константой или переменной, а также указывать на константу или переменную

Примеры:

```
int i;          // целая переменная
const int ci =1;     // целая константа
int *pi;        // указатель на целую переменную
const int *pci;  // указатель на целую константу
```

```
    int * const cp =&i;
/*указатель-константа на целую переменную */

    const int * const cpc = &ci;
/* указатель-константа на целую константу */
```

- *модификатор **const**, находящийся между именем указателя и звездочкой, относится к самому указателю и запрещает его изменение*
- ***const** слева от звездочки задает постоянство значения, на которое он указывает.*
- *для инициализации указателей использована операция получения адреса &.*

Инициализация указателей

- При определении указателя надо стремиться выполнить его инициализацию, то есть присвоение начального значения.

Непреднамеренное использование неинициализированных указателей — распространенный источник ошибок в программах.

- Существуют следующие способы инициализации указателя:
  - 1. Присваивание указателю адреса существующего объекта
  - 2. Присваивание указателю адреса области памяти в явном виде
  - 3. Присваивание пустого значения
  - 4. Выделение участка динамической памяти и присваивание ее адреса указателю

Инициализация указателей

- 1. Присваивание указателю адреса существующего объекта:
  - с помощью операции получения адреса:
  - **int a = 5; // целая переменная**
  - **int* p = &a; // в указатель записывается адрес a**

  - с помощью значения другого инициализированного указателя:

    int* r = p;

  - с помощью имени массива или функции, которые трактуются как адрес:
  - **int b[10]; // массив**
  - **int* t = b; // присваивание адреса начала массива**
  - **….**
  - **void f(int a){ /* … */ } // определение функции**
  - **void (*pf)(int); // указатель на функцию**
  - **pf = f; // присваивание адреса функции**

Способы инициализации указателей

- 2. Присваивание указателю адреса области памяти в явном виде:
  - **char* vp = (char *) 0xB8000000;**
- 3. Присваивание пустого значения:
  - **int *suxx = NULL;**
  - **int *rulez = 0;**

Способы инициализации указателей

- 4. Выделение участка динамической памяти и присваивание ее адреса указателю:
  - с помощью операции **new**:
  - **int *n = new int;** **// 1**
  - **int *m = new int (10);** **// 2**
  - **int *q = new int [10];** **// 3**

  - с помощью функции **malloc**:
  - **int *u = (int *)malloc(sizeof(int));** **// 4**

В операторе 1 выделяется участок динамической памяти для размещения величины типа **int** и записывает адрес начала этого участка в переменную **n**.

В операторе 2 кроме описанных выше действий, производится инициализация выделенной динамической памяти значением 10.

В операторе 3 операция **new** выполняет выделение памяти под 10 величин типа **int** и записывает адрес начала этого участка в переменную **q**, которая может трактоваться как имя массива.

В операторе 4 делается то же самое, что и в операторе 1, но с помощью функции выделения памяти **malloc**, унаследованной из библиотеки C.

Способы инициализации указателей

Некоторые замечания:

Память под сам указатель выделяется на этапе компиляции.

Если память выделить не удалось, по стандарту должно порождаться исключение **bad_alloc**.

- старые версии компиляторов могут возвращать 0.

Операцию **new** использовать предпочтительнее, чем функцию **malloc.**

- особенно при работе с объектами.

Для того чтобы использовать **malloc**, требуется подключить к программе заголовочный файл **<malloc.h>**.

С помощью комбинаций звездочек, круглых и квадратных скобок можно описывать составные типы и указатели на составные типы

Например: **int *(*p[10])();**

объявляется массив из 10 указателей на функции без параметров, возвращающих указатели на **int**.

При интерпретации сложных описаний необходимо придерживаться правила ***«изнутри наружу»:***

1

2

3

4

Способы инициализации указателей

Для описания

int *(*p[10])();

порядок интерпретации указан цифрами:

```
int    * (*p [10]) ();
 5     4  2    1     3
```

По умолчанию квадратные и круглые скобки имеют одинаковый приоритет, больший, чем звездочка, и рассматриваются слева направо.

Для изменения порядка рассмотрения используются круглые скобки.

Способы инициализации указателей

Освобождение памяти, выделенной с помощью операции **new**, должно выполняться с помощью **delete**, а памяти, выделенной функцией **malloc** — посредством функции **free**.

- **delete n; delete m; delete [] q; free (u);**

Если память выделялась с помощью **new[]**, для освобождения памяти необходимо применять **delete[]**.
Размерность массива при этом не указывается.

- Если квадратных скобок нет, то сообщение об ошибке не выдается, но помечен как свободный будет только первый элемент массива, а остальные окажутся недоступны для дальнейших операций.
- Такие ячейки памяти называются мусором.

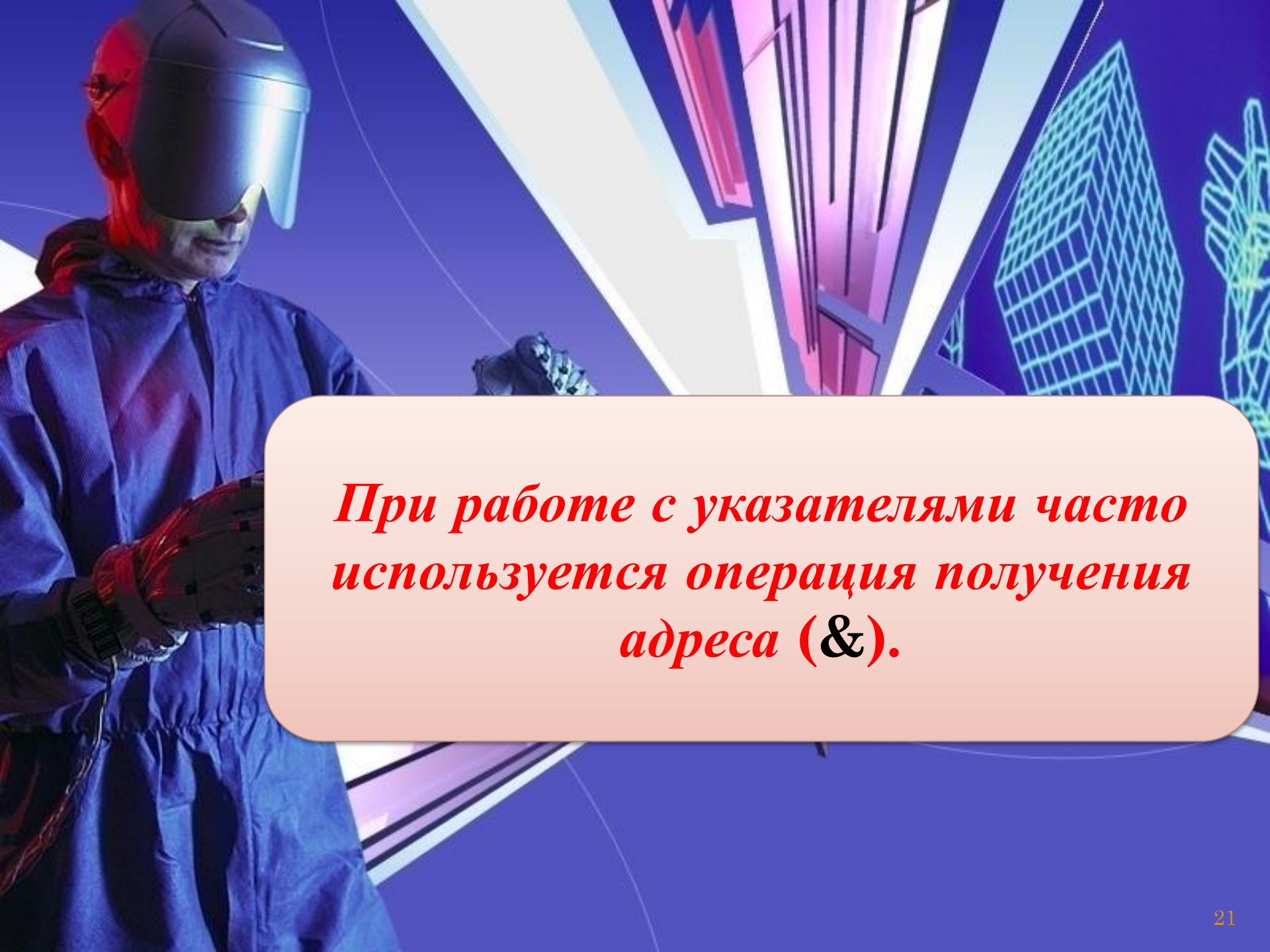
Если переменная-указатель выходит из области своего действия, отведенная под нее память освобождается.

- Следовательно, динамическая переменная, на которую ссылался указатель, становится недоступной.
- При этом память из-под самой динамической переменной не освобождается.

Освобождение памяти, выделенной под указатели

Операции с указателями

- С указателями можно выполнять следующие операции:
  - разадресация, или косвенное обращение к объекту (*)
  - присваивание
  - сложение с константой
  - вычитание
  - инкремент (++)
  - декремент (--)
  - сравнение
  - приведение типов



При работе с указателями часто используется операция получения адреса (&).

Операция разадресации.

предназначена для доступа к величине, адрес которой хранится в указателе

```
char a;                  // переменная типа char

char *p = new char;      /* выделение памяти под указатель и под
                            динамическую переменную типа char */

*p = 'Ю'; a = *p;        // присваивание значения обеим переменным
```

На одну и ту же область памяти может ссылаться несколько указателей различного типа.

Примененная к ним операция разадресации даст разные результаты

Например:

```
#include <stdio.h>
int main(){
    unsigned long int A = 0Xcc77ffaa;
    unsigned short int* pint = (unsigned short int*) &A;
    unsigned char* pchar = (unsigned char *) &A;
    printf("| %x | %x | %x |", A, *pint, *pchar);
    return 0; }
```

на IBM PC-совместимом компьютере будет выведена на экран строка:

| cc77ffaa | ffaa | aa |

Арифметические операции с указателями (сложение с константой, вычитание, инкремент и декремент) автоматически учитывают размер типа величин, адресуемых указателями.

Эти операции применимы только к указателям одного типа и имеют смысл в основном при работе со структурами данных, последовательно размещенными в памяти, например, с массивами.

Арифметические операции с указателями

Инкремент

- перемещает указатель к следующему элементу массива

Декремент

- перемещает указатель к предыдущему элементу массива

Фактически значение указателя изменяется на величину **sizeof(тип)**.

Если указатель на определенный тип увеличивается или уменьшается на константу, его значение изменяется на величину этой константы, умноженную на размер объекта данного типа

```
short * p = new short [5];
p++;                          // значение p увеличивается на 2
long * q = new long [5];
q++;                          // значение q увеличивается на 4
```

Арифметические операции с указателями

Разность

- разность двух указателей — это разность их значений, деленная на размер типа в байтах

в применении к массивам разность указателей, например, на третий и шестой элементы равна 3

Суммирование двух указателей не допускается!

Ссылки

ССЫЛКИ.

- *Для удобства записи программного кода и улучшения читаемости программы в языке C определено понятие ссылки.*
- *Ссылка представляет собой синоним имени, указанного при инициализации ссылки.*
- *Ссылку можно рассматривать как указатель, который всегда разыменовывается.*

Формат объявления ссылки:

тип & имя;

где

тип — *это тип величины, на которую указывает ссылка.*

& — *оператор ссылки, означающий, что следующее за ним* **имя** *является именем переменной ссылочного типа.*

Ссылки применяются чаще всего в качестве параметров функций и типов возвращаемых функциями значений.

- Ссылки позволяют использовать в функциях переменные, передаваемые по адресу, без операции разадресации, что улучшает читаемость программы.

Ссылка, в отличие от указателя,
не занимает дополнительного пространства в памяти
и является просто другим именем величины.

Операция над ссылкой приводит к изменению величины,
на которую она ссылается.

Примеры :

```
int kol;

int & pal = kol;    // ссылка pal - альтернативное имя для kol

const char & CR = '\n';     // ссылка на константу
```

Запомните следующие правила:

Переменная-ссылка должна явно инициализироваться при ее описании

- кроме случаев, когда она является параметром функции, описана как **extern** или ссылается на поле данных класса.

После инициализации ссылке не может быть присвоена другая переменная

Тип ссылки должен совпадать с типом величины, на которую она ссылается

Не разрешается определять указатели на ссылки, создавать массивы ссылок и ссылки на ссылки