**1. Для чего используется указатель на void? Приведите примеры.**

Указатель void\* - бестиповой указатель, указывает на произвольное место в памяти.

Тип указателя void\* используется, если тип объекта неизвестен.

Пример:

void swap (void \*a, void \*b, size\_t size)

{

char\* tmp;

tmp = (char\*) malloc(size);

memcpy (tmp, a, size);

memcpy (a, b, size);

memcpy (b, tmp, size);

free(tmp);

}

**2. Каковы особенности использования указателя на void? Приведите примеры.**

- Позволяет передавать в функцию указатель на объект любого типа

void swap (void \*a, void \*b, size\_t size)

- Полезен для ссылки на произвольный участок памяти, независимо от размещенных там объектов.

- Указателя типа void\* нельзя разыменовывать.

int a = 10;

int \*pi = &a;

void \*pv = pi;

\*pv; // Ошибка – нельзя разыменовывать void\*

- К указателям типа void\* не применима адресная арифметика

- В языке С допускается присваивание указателя типа void указателю любого другого типа (и наоборот) без явного преобразования типа указателя.

double d = 5.0;

double \*pd = &d;

void \*pv = pd;

pd = pv;

**16. Указателя на функцию и указатель на void.**

Указатель на функцию представляет собой выражение или переменную, которые используются для представления адреса функции. Указатель на функцию содержит адрес первого байта в памяти, по которому располагается выполняемый код функции.

тип (\*имя\_указателя) (параметры);

Здесь **тип** представляет тип возвращаемого функцией значения.

**имя\_указателя** представляет произвольно выбранный идентификатор в соответствии с правилами о наименовании переменных.

И **параметры** определяют тип параметров через запятую при их наличии.

Указатель void\* - бестиповой указатель, указывает на произвольное место в памяти.

Указатель на функцию не может быть преобразован к указателю на void и наоборот

**11. Для чего используется указатель на функцию? Приведите примеры.**

**-** Объявление указателя на функцию

double trapezium (double a, double b, int n, double (\*func) (double));

- Получение адреса функции

result = trapezium(0, 3.14, 25, &sin);

- Вызов функции по указателю

y = (\*func) (x); // y = func(x);

**12. Указатель на функцию: описание, инициализация, вызов функции по указателю.**

Указатель на функцию представляет собой выражение или переменную, которые используются для представления адреса функции. Указатель на функцию содержит адрес первого байта в памяти, по которому располагается выполняемый код функции.

Инициализация

тип (\*имя\_указателя) (параметры);

Вызов функции по указателю

y = (\*func) (x); // y = func(x);

**13. Функция qsort, примеры использования.**

void qsort (void \*base, size\_t num, size\_t size, int (\*compare) (const void\*, const void\*));

Функция qsort() сортирует массив, на который указывает параметр **base**. Параметр **num** задает число элементов массива, параметр **size** задает размер в байтах каждого элемента.

Функция, на которую указывает параметр compare, сравнивает элементы массива с ключом. Формат функции compare следующий:

int func\_name(const void \*arg1, const void \*arg2)

Она должна возвращать следующие значения:

- Если arg1 меньше, чем arg2, то возвращается значение меньше 0.

- Если arg1 равно arg2, то возвращается 0.

- Если arg1 больше, чем arg2, то возвращается величина больше 0.

Массив сортируется по возрастанию таким образом, что наименьший адрес соответствует наименьшему элементу.

Пример: (сам)

**14. Особенности использования указателей на функцию.**

- Выражение из имени функции неявно преобразуется в указатель на функцию.

int add (int a, int b);

...

int (\*p1) (int, int) = add;

- Операция "&" для функции возвращает указатель на функцию.

int (\*p2) (int, int) = &add;

- Операция "\*" для указателя на функцию возвращает саму функцию, которая неявно преобразуется в указатель на функцию.

int (\*p3) (int, int) = \*add;

int (\*p4) (int, int) = \*\*\*\*\*add;

- Указатели на функции можно сравнивать

if (p1 == add)

printf ("p1 points to add\n");

- Указатель на функцию может быть типом возвращаемого значения функции.

**15. Указатель на функцию и адресная арифметика.**

**3. Функции для выделения и освобождения памяти malloc, calloc, free. Порядок работы и особенности использования этих функций.**

- malloc (size\_t size) (выделяет блок памяти и не инициализирует его);

- calloc (size\_t num, size\_t size) (выделяет блок памяти и заполнят его нулями)

- free (void \*ptr) (освобождает ранее выделенный блок памяти, на который указывает ptr)

Особенности использования:

- Указанные функции не создают переменную, они лишь выделяют область памяти. В качестве результата функции возвращают адрес расположения этой области в памяти компьютера, т.е. указатель.

- Поскольку ни одна из этих функций не знает данные какого типа будут располагаться в выделенном блоке все они возвращают указатель

на void.

- В случае если запрашиваемый блок памяти выделить не удалось, любая из этих функций вернет значение NULL.

- После использования блока памяти он должен быть освобожден. Сделать это можно с помощью функции free.

**4. Функция realloc. Особенности использования.**

Функция realloc (void \*ptr, size\_t size) перевыделяет предварительно выделенный блок памяти.

ptr == NULL && size != 0 -> Выделение памяти (как malloc)

ptr != NULL && size == 0 -> Освобождение памяти (как free).

ptr != NULL && size != 0 -> Перевыделение памяти.

В худшем случае:

• выделить новую область

• скопировать данные из старой области в новую

• освободить старую область

**5. Общие «свойства» функций malloc, calloc, realloc**

- Указанные функции не создают переменную, они лишь выделяют область памяти. В качестве результата функции возвращают адрес расположения этой области в памяти компьютера, т.е. указатель.

- Поскольку ни одна из этих функций не знает данные какого типа будут располагаться в выделенном блоке все они возвращают указатель

на void.

- В случае если запрашиваемый блок памяти выделить не удалось, любая из этих функций вернет значение NULL.

- После использования блока памяти он должен быть освобожден. Сделать это можно с помощью функции free.

**6. Функция выделения памяти и явное приведение типа: за и против.**

Функция выделения памяти: malloc, calloc

Преимущества явного приведения типа:

– компиляции с помощью c++ компилятора;

– у функции malloc до стандарта ANSI C был другой прототип (char\* malloc(size\_t size));

– дополнительная «проверка» аргументов разработчиком.

Недостатки явного приведения типа:

– начиная с ANSI C приведение не нужно;

– может скрыть ошибку, если забыли подключить stdlib.h;

– в случае изменения типа указателя придется менять и тип в приведении.

**7. Особенности выделения 0 байт памяти.**

Результат вызова функций malloc, calloc или realloc, когда запрашиваемый размер блока равен 0, зависит от реализации

– вернется нулевой указатель;

– вернется «нормальный» указатель, но его нельзя использоваться для разыменования.

ПОЭТОМУ перед вызовом этих функций нужно убедиться, что запрашиваемый размер блока не равен нулю.

**8. Способы возвращения динамического массива из функции.**

- Как возвращаемое значение

int\* create\_array(FILE \*f, int \*n);

int \*arr, n;

arr = create\_array(f, &n);

- Как параметр функции

int create\_array(FILE \*f, int \*\*arr, int \*n);

int \*arr, n, rc;

rc = create\_array(f, &arr, &n);

**9. Типичные ошибки при работе с динамической памятью (классификация, примеры)**

**•** Неверный расчет количества выделяемой памяти.

• Отсутствие проверки успешности выделения памяти

• Утечки памяти

• Логические ошибки

- Использование непроинициализированного указателя.

- Использование указателя сразу после освобождения памяти.

- Изменение указателя, который вернула функция выделения памяти.

- Двойное освобождение памяти.

- Освобождение невыделенной или нединамической памяти.

- Выход за границы динамического массива.

- И многое другое

**10. Подходы к обработке ситуации отсутствия свободной памяти при выделении.**

• Возвращение ошибки (англ., return failure)

– Подход, который используем мы

• Ошибка сегментации (англ., segfault)

– Обратная сторона - проблемы с безопасностью

• Аварийное завершение (англ., abort)

– Идея принадлежит Кернигану и Ритчи (xmalloc)

• Восстановление (англ., recovery)

– xmalloc из git

**17. Утилита make: назначение, входные данные, идея алгоритма работы.**

make — утилита, автоматизирующая процесс преобразования файлов из одной формы в другую.