Пусть есть функция f.

void f() {}

void (\*pf) () = &f; // запись означает объявить переменную pf типа указатель на функцию, которая возвращает void и не принимает параметров. Мы к этому указателю присваиваем адрес f; Второй вариант синтаксиса void (\*pf) () = f; // адрес места в памяти, где находится код функции f

pf(); вызов функции по указателю

Ещё пример:

void f (int a) {}

void g (int b) {}

void (\*pf) (int) = &f;

pf(10);

pf=&g;

pf(20);

Здесь может применяться любая сложная логика, выбор функции может зависеть от введённого пользователем значения, от случайного числа. Выбор реального пути выполнения происходит не в момент компиляции, а в момент запуска и выполнения собственно.

Как это реализовано? Стоить посмотреть .asm листинг.

Ещё примеры: qsort.

traverse (tree\*) :

Пусть есть структура Node и на её основе создано дерево Tree. Есть функция обхода этого дерева. И мне как разработчику такого дерева и такого метода хочется предоставить пользователю способ выполнения для каждого узла дерева какой-то обработки, вызова какой-то функции. То есть обход – не единственная задача моего метода, надо ещё выполнить что-то в каждом Node.

struct Node { ... }; struct Tree { Node \*head; ... };

void traverse (Tree \*t, void (\*do)(Node \*n)) // второй параметр – функция которая в качестве своего параметра берёт узел

{

.... обход в глубину....

в каждой вершине do(v);

}

На момент разработки библиотеки не известно, что пользователи будут делать при обходе. В качестве функции do пользователи могут определять:

void print (Node\* n) { вывод чего-то из n }

void calc (Node\* n) { подсчет числа каких-то особенных узлов }

соответственно можно вызывать traverse(t,print); или traverse(t,calc);

Указатели на функцию можно объединить в структуру.

struct matrix\_operations {

void (\*mult) (int\*\*, int\*\*, int\*\*)

void (\*add) (int\*\*, int\*\*, int\*\*)

};

Это может потребоваться.

Например, пусть есть большой кусок кода, производящий математические действия с векторами, перемножает, складывает ... Современные процессоры содержат набор инструкций, ускоряющий такие операции, в том числе позволяющие векторизовать операции однотипные. Это MMX, SSE, SSE2 и т.д. наборы команд. Точно также и с технологиями CUDA и Firestream.

Программа стартует и проверяет поддержку набора команд.

operation op;

if (MMX) op.add = add\_MMX; // эффективно реализованная в ASM инструкциях быстрая функция

else op.add = add\_SIMPLE;

а дальше во все функции полезного куска кода передаём &op; Внутри вызываем op->add, op->mult и так далее. Этот большой полезный кусок кода мог быть написан тогда, когда ещё не было никакого MMX или SSE2, но разработчик предусмотрел, что могут появится быстрые реализации функций и предпочел работать со структурой содержащей адреса функций.

Последний пример callback. Функции для обратного вызова. Есть разные графические библиотеки например MFC -> Winforms, QT, есть написанная на си GTK (GNOME написана с использованием GTK). Как работают большинство gui программ?

main () {

gtk\_start(); // всегда будет какая-нибудь инструкция – граф. библиотека начни работать, в этот момент управление передаётся графической системе, пока вы окно не закроете управление висит. Внутри граф. системы как правило есть очередь событий. Пусть это будет массив, в который ОС складывает информацию о том, какие действия пользователь произвел : нажал клавишу, нажал кнопку мыши, поводил мышью, свернул или развернул окно. Граф. библиотека крутится в бесконечном цикле проверяя события. Я, например, хочу, чтобы моя функция вызывалась, когда пользователь нажал мышкой. Как это происходит? Сообщить граф. системе о том, какую функцию надо вызывать при событии можно с помощью указателей на функции, обычно это так и делается. Пусть есть моя функция f (int x, int y), я сообщаю граф. библиотеке gtk\_set\_mouse\_handler (f); Граф. система понимает, если произошло такое событие, то нужно вызывать функцию f. Это и есть callback-функция, то есть я передаю указатель на функцию, а когда происходит соответствующее событие эту функцию вызывают. Идеологически это опять использование указателей на функцию, но настолько важное, что применяется очень часто.