Язык Си

Организация кода

1. Включение заголовочных файлов
2. Макроопределения, макроподстановки
3. Определение глобальных переменных
4. Описание прототипов функций
5. Определение функций.
6. Точка входа (`main`), если необходимо

Типы данных

Язык Си является языком со статической типизацией, т.е. тип переменной известен на этапе компиляции и не меняется в процессе работы. Компилятор берет на себя проверку типов, снимая с программиста заботу о проверке типов(в отличии от динамической типизации где на этапе исполнения типы переменных могут не совпадать и произойдет ошибка процесса исполнения). В современных языках используется прием вычисление типа на этапе компиляции. Если переменная обозначена специальным типом, то в процесс екомпиляции будет расичтан тип переменной и назначен ей. Такой прием применяется в новых стандартах языка `C++`, вычисляемым типом является `auto`.

Также, в языка Си применется слабая типизация, т.е. есть возможность привести один тип к другому, без проверки. Данный

прием называется приведением типов. Эта операция считается опасной, и может привести к ошибкам на этапе исполнения программы, если типы не совпадают. Например при не совпадении типов указателей, при итерировании, можно выйти за границу массива и повредить память или при приведении типа `long long` к типу `int` можно потерять часть данных которые выходят за размер переменной. В современных компиляторах, преобразование типов, на стадии семантического анализа, производится попытка предположить возможность появления ошибки. Приведение типов записывается как `(int *)v` - что означает приведение переменной `v` к типу `int *` и назвается `C-style` преобразование типов. Данный вид записи встречается во многих языках, нгапример в `Java`, `C#` - хотя в нем есть более удачные конструкции преобразования. Из-за приведения типов, язык не может считаться типобезопасным. В `C++`, такой стиль считается устаревшим и выдается предупреждение на этапе компиляции и предлагается использовать `static_cast`, `const_cast`, `reinterpret_cast`, `dynamic_cast` - что является более безопасным и контролируемым компилятором.

Немного отступая, можно рассказать еще о так называемой “утиной типизации” - это такой вид типизации когда интерфейс явно не реализуется, а реализуются его методы. Данные подход применяется в языке `GoLang`.

Си также не грешит “каламбуром” типизации. Это когда обходится проверка системы типов компилятора, для выполнения определенных задач. Одним из примеров может быть интерфейс сокетов. Функция `bind` имеет следующее описание:

```
int bind(int sockfd, struct sockaddr *addr, socklen_t addrlen
);
```

Вызов функции происходит примерно следующим образом

```
struct sockaddr_in sa = {0};
int sockfd = ...;
sa.sin_family = AF_INET;
sa.sin_port = htons(port);
bind(sockfd, (struct sockaddr *)&sa, sizeof sa);
```

Применяется за основу тот факт, что в языке указатель на `struct sockaddr_in` может беспрепятственно преобразовываться в указатель на `struct sockaddr`, а также что оба структурных типа частично совпадают по организации представления в памяти. Следовательно, указатель на поле `addr->sin_family` (где `addr` имеет тип `struct sockaddr*`) на самом деле будет указывать на поле `sa.sin_family` (где `sa` имеет тип `struct sockaddr_in`). Другими словами, библиотека использует каламбур типизации для реализации примитивной формы наследования.

Такой же подход можно встретить в `Windows API`.

Примеры типизации в других языках(`Asm` , `C++` , `Erlang` , `C#` , `Forth`)

Тип	Размер	Диапазон

char	1	
short	2	
int	4	
long	4	
float	4	
double	8	

Макроподстановки.

Макроопределения.

Структуры. Перечисления.

Объединения.

```
struct Person {
  char *first_name;
  char *last_name;
  int   age;
};
```

```
enum Direct {
  Left, Right, Unknown = -1
};
```

```
union Match {
```

```
  struct Binary {
    unsigned char bin[4];
  } binary;
  unsigned long uni;
};
```

Строки

```
char *str = 0;
```

Массивы

Переменные

```
[модификаторы] тип имя [инициализация]
```

Блок кода. Область видимости

```
{}
```

Операции

Операция - это некоторая функция, которая выполняется над операндами и которая возвращает вычисленное значение - результат выполнения операции.

Унарные операции

Унарные операции - это операции, содержащие единственный операнд.

К унарным операциям в Си относятся следующие операции:

1. `+` (унарный плюс),
2. `-` (унарный минус),
3. `~` (взятие обратного кода),
4. `!` (логическое отрицание),
5. `&` (взятие адреса),
6. `*` (операция разыменовывания указателя),
7. `sizeof` (операция определения занимаемого объектом объёма памяти).

Бинарные операции

Бинарные операции - это операции, содержащие два операнда, между которыми расположен знак операции.

К бинарным операциям в Си относятся следующие операции:

1. `+` (сложение),
2. `-` (вычитание),
3. `*` (умножение),
4. `/` (деление),
5. `%` (взятие остатка от деления),
6. `&` (поразрядное И),
7. `|` (поразрядное ИЛИ),

8. `^` (поразрядное исключающее ИЛИ),
9. `<<` (логический сдвиг влево),
10. `>>` (логический сдвиг вправо),
11. `&&` (логическое И),
12. `||` (логическое ИЛИ).

Также к бинарным операциям в Си относятся операции, по сути представляющие собою присваивание:

1. `+=` (добавление к левому операнду значения, представленного правым операндом);
2. `-=` (вычитание из левого операнда значения, представленного правым операндом);
3. `*=` (умножение левого операнда на значение, представленное правым операндом);
4. `/=` (деление левого операнда на значение, представленное правым операндом);
5. `&=` (поразрядное логическое И над левым и правым операндом);
6. `|=` (поразрядное логическое ИЛИ над левым и правым аргументом);
7. `^=` (поразрядное логическое исключающее ИЛИ над левым и правым аргументом);
8. `<<=` (поразрядный сдвиг влево левого аргумента на количество бит, заданное правым аргументом);
9. `>>=` (поразрядный сдвиг вправо левого аргумента на количество бит, заданное правым аргументом).
   Данные операции предполагают, что левый операнд

представляет собою лево-допустимое выражение.

Тернарные операции

В Си имеется единственная тернарная операция - условная операция, которая имеет следующий вид:

`[условие]? [выражение1] : [выражение2] ;`
и которая имеет три операнда:

- `[условие]` - логическое условие, которое проверяется на истинность,
- `[выражение1]` - выражение, значение которого возвращается в качестве результата выполнения операции, если условие истинно;
- `[выражение2]` - выражение, значение которого возвращается в качестве результата выполнения операции, если условие ложно.
  Знаком операции здесь служит целое сочетание `?` `:`.

Функции и типы вызовов. Адрес возврата. Таблица функций.

Функция - это самостоятельный фрагмент программного кода, который может многократно использоваться в программе. Функции могут иметь аргументы и могут возвращать значения.

Для того, чтобы задать функцию в Си, необходимо её объявить:

- сообщить имя (идентификатор) функции,
- перечислить входные параметры (аргументы)
- указать тип возвращаемого значения,

  Также необходимо привести определение функции, которое содержит блок операторов, реализующих поведение функции.

Отсутствие определения ранее определённой функции является ошибкой, что, в зависимости от реализации, приводит к выдаче сообщений или предупреждений.

Когда компилятор встречает в программном коде идентификатор функции, то он оформляет операцию вызова функции, в рамках которой, в частности, адрес точки вызова помещается в стек, создаются и инициализируются переменные, отвечающие за параметры функции, и передаётся управление коду, реализующему вызываемую функцию. После выполнения функции происходит освобождение памяти, выделенной при вызове функции, возврат в точку вызова и, если вызов функции является частью некоторого выражения, передача в точку возврата вычисленного внутри функции значения.

Особый класс функций представляют встраиваемые (или подставляемые) функции - функции, объявленные с указанием ключевого слова `inline`. Определения таких функций непосредственно подставляются в точку вызова, что, с одной стороны, увеличивает объём исполняемого кода, но, с другой стороны, позволяет экономить время его выполнения, поскольку не используется дорогая по времени операция вызова функции.

Объявление функции

Объявление функции имеет следующий формат:

```
[описатель] [имя] ( [список] );,
```

где

- `[описатель]` - описатель типа возвращаемого функцией значения;
- `[имя]` - имя функции (уникальный идентификатор функции);
- `[список]` - список (формальных) параметров функции. Признаком объявления функции является символ `;`, таким образом, объявление функции - это инструкция.

В самом простом случае `[описатель]` содержит указание на конкретный тип возвращаемого значения. Функция, которая не должна возвращать никакого значения, объявляется как имеющая тип `void`. При необходимости в описателе могут присутствовать дополнительные элементы:

- модификатор `extern` указывает на то, что определение функции находится в другом модуле;
- модификатор `static` задаёт статическую функцию;
- модификаторы `pascal` или `cdecl` влияют на обработку формальных параметров и связаны с подключением внешних модулей.

Список параметров функции задаёт сигнатуру функции.

Си не допускает объявление нескольких функций, имеющих одно и то же имя, перегрузка функций не поддерживается.

Определение функции

Определение функции имеет следующий формат:

```
[описатель] [имя] ( [список] ) [тело]
```

Где `[описатель]`, `[имя]` и `[список]` - те же, что и в объявлении, а `[тело]` - это составной оператор, который представляет собою конкретную реализацию функции. Компилятор различает определения одноимённых функций по их сигнатуре, и таким образом (по сигнатуре) устанавливается связь между определением и соответствующим ему объявлением.

Тело функции имеет следующий вид:

```
{
   [последовательность операторов]
   return ([возвращаемое значение]) ;
}
```

Вызов функции

Вызов функции заключается в выполнении следующих действий:

- сохранение точки вызова в стеке;
- выделение памяти под переменные, соответствующие формальным параметрам функции;
- инициализация переменных значениями переменных (фактических параметров функции), переданных в функцию при её вызове, а также инициализация тех переменных, для которых в объявлении функции указаны значения по умолчанию, но для которых при вызове не были указаны соответствующие им фактические параметры;
- передача управления в тело функции.

В зависимости от реализации, компилятор либо строго следит за тем, чтобы тип фактического параметра совпадал с типом формального параметра, либо, если существует такая возможность, осуществляет неявное преобразование типа, что, очевидно, приводит к побочным эффектам.

Если в функцию передаётся переменная, то при вызове функции создаётся её копия (в стеке выделяется память и копируется значение). Например, передача структуры в функцию вызовет копирование всей структуры целиком. Если же передаётся указатель на структуру, то копируется только значение указателя. Передача в функцию массива также вызывает лишь копирование указателя на его первый элемент. При этом для явного обозначения того, что на вход функции принимается адрес начала массива, а не указатель на единичную переменную, вместо объявления указателя после названия

переменной можно поставить квадратные скобки, например:

```
void example_func(int *array);
```

Си не допускает вложенные вызовы.

Частный случай вложенного вызова - это вызов функции внутри тела вызываемой функции. Такой вызов называется рекурсивным, и применяется для организации единообразных вычислений. Учитывая естественное ограничение на вложенные вызовы, рекурсивную реализацию заменяют на реализацию при помощи циклов.

Возврат из функции

При возврате из функции освобождается память, выделенная под параметры функции и под переменные, объявленные внутри функции, и управление возвращается в точку вызова.

Конструкции языка