**История разработки языка С**

Язык "С" был создан Денисом Ричи в 1972 году. Он создавался для разработки операционной системы UNIX. Существует широкий спектр языков программирования высокого уровня: С, Perl, Basic, Java, C# (Си – шарп), приведем причины по которым язык С ставится на первое место:

1. С – язык мощный и гибкий. Не имеет ограничений. Используется при разработке ОС, текстовых редакторов, графических приложений, компиляторов других языков.
2. С – хорошо переносимый язык. Т.е. программу, написанную для одной компьютерной системы можно скомпилировать и запустить в другой среде без изменений.
3. С – очень краткий язык.
4. С – модульный язык. Программу на С можно писать в виде набора подпрограмм-модулей, которые называются функциями.

На базе языка С были созданы следующие языки объектно-ориентированного программирования: С++, Java, С#.

**Цикл разработки программы на языке С**

Существуют свои этапы разработки программы. На первом этапе с помощью текстового редактора создается дисковый файл, содержащий исходный текст(код) программы. На втором этапе исходный код компилируется и создается объектный файл. На третьем этапе скомпилированный код обрабатывается компоновщиком для получения исполняемого файла. На четвертом этапе программа запускается на выполнение.

Создание исходного кода программы

***Исходный код –*** последовательность команд. Большинство компилятором снабжены встроенным текстовым редактором. Но при его отсутствии можно воспользоваться любым текстовым редактором. Notepad, WordPad (from Windows), Edit (MS DOS), либо сохранять файлы исходного кода в формате ASCII используя редакторы WordPerfect, MS Word и другие. При сохранении файла ему присваивают расширение **.с**

Например: **radius.c**

Компиляция исходного кода

Перевод исходного кода на машинный язык называется ***компиляцией.*** Последовательность инструкций на машинном языке, сгенерированная компилятором называется объектным кодом, а дисковый файл, содержащий её – объектным файлом. Существуют различные компиляторы для различных ОС, а следовательно каждому компилятору необходима своя команда.

|  |  |
| --- | --- |
| Компилятор | Команда |
| Microsoft C  Borland Turbo C  Borland C  GNU C/C++ | cl radius.c  tcc radius.c  bcc radius.c  gcc radius.c |

Для OS UNIX команда **cc radius.c**

Компиляция дает объектный файл с расширением **.obj** (в системах UNIX и Linux расширение **.о**)

Существуют графические интегрированные среды разработки программ (Bloodshed Dev-C++, C++ Builder 5)

Создание исполняемого файла

Компоновщик объектный файл соединяет с библиотеками функций для получения ***исполняемого файла (.exe).*** После того как получен исполняемый файл, можно запускать его на выполнение. В некоторых редакторах компоновка и компиляция осуществляются вместе.

Первая программа на языке С

**#include <stdio.h>**

**int main(void)**

**{**

**printf("Hello World!\n");**

**return 0;**

**}**

В результате на экране появится фраза: **Hello World!**

Разберём приведенную программу. Директива **#include** указывает компилятору С содержимое какого заголовочного файла необходимо включить в программу во время компиляции. ***Заголовочный файл –*** это отдельный файл на диске, содержащий информацию необходимую программе. Существует несколько заголовочный файлов (stdio.h, calc.h, stdlib.h и другие). Функция **main** это единственный элемент, который обязательно должен присутствовать в программе. В простом случае эта функция состоит из имени **main** за которым следует **(void)**, а затем {}. **Void** означает что в функцию не передаются никакие данные. Можно просто писать **main().** В {} стоят операторы тела программы. Оператор **return** обязательно должен присутствовать для осуществления выхода из функции. **Printf –** вызов библиотечной функции, которая осуществляет вывод информации и данных на экран.

**Алфавит языка С**

Для записи конструкций в языке С используются:

1. Заглавные и строчные буквы (причём малые отличаются от больших). Для записи имен может использоваться знак \_подчеркивания, который эквивалентен букве.
2. Цифры 0..9
3. Специальные символы ',".
4. Группа основных символов – набор служебных слов. Они зарезервированы и не могут использоваться в других целях

**asm**

**auto**

**break**

**case**

**char**

**const**

**continue**

**default**

**do**

**double**

**else**

**enum**

**extern**

**float**

**goto**

**if**

**inline**

**int**

**long**

**register**

**restrict**

**return**

**short**

**signed**

**sizeof**

**static**

**struct**

**switch**

**typedef**

**union**

**unsigned**

**void**

**volatile**

**while**

В конкретных реализациях языка имеется ряд дополнительных слов.

**Типы числовых переменных**

В языке С существуют только целочисленные переменные и переменные с плавающей точкой для хранения вещественных и действительных чисел.

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Название** | **Требует байт** | **Диапазон** |
| Символ | **Char** | 1 | -128..127 |
| Короткое целое | **Short** | 2 | -32 767..32 767 |
| Целое | **Int** | 4 | -2 147 483 647.. 2 147 483 647 |
| Длинное целое | **Long** | 4 | -2 147 483 647.. 2 147 483 647 |
| Очень Длинное целое | **Long long** | 8 | -9 223 372 036 854 775 807..  9 223 372 036 854 775 807 |
| Символ без знака | **Unsigned char** | 1 | 0.. 255 |
| Короткое целое без знака | **Unsigned short** | 2 | 0.. 65 535 |
| целое без знака | **Unsigned int** | 4 | 0.. 4 294 967 295 |
| Длинное целое без знака | **Unsigned long** | 4 | 0.. 4 294 967 295 |
| Очень Длинное целое без знака | **Unsigned long long** | 8 | 0.. 18 446 744 073 709 551 615 |
| Число с плавающей точкой один. точности | **Float** | 4 | 1.2E-38..3.4E38  (точность 7 цифр) |
| Число с плавающей точкой двойной точности | **Double** | 8 | 2.2Е-308..1.8Е308  (точность 19 цифр) |

**Объявление переменных**

Итак, отдельная единица данных должна обязательно иметь определенный тип. Для ее хранения во время работы программы мы должны, во-первых, отвести соответствующее место в памяти, а во-вторых, идентифицировать ее, присвоив некоторое имя. Именованная единица памяти для хранения данных называется ***переменной***. Переменные создаются с помощью оператора объявления переменных, в котором указывается тип, имена переменных и (при необходимости) начальные значения, которыми переменные инициализируются. Переменную необходимо объявить до её использования в программе. Объявление переменных имеет следующий вид:

**Тип имя;**

int a1, b1, d=0;

float x, y;

char c, a[20];

char s='a';

В программе могут использоваться массивы, которые описываются аналогично простым переменным, но за именем массива указывается максимальное значение его индексов. Все переменные лучше объявлять перед функцией **main()**, но бывают и исключения.

Имеется возможность изменить имя существующего типа, для этого имеется функция **typedef:**

Например: typedef int integer;

После этого можно пользоваться словом integer для объявления переменных типа int.

**Константы**

***Константа*** – это именованный участок памяти, используемый для хранения данных. Значения константы не изменяется. Существуют ***литерные константы*** и ***символические константы***.

Литерные константы

Это литерал – значение, стоящее в сходном коде программы непосредственно там, где используется. Эти литералы могут быть числовыми, символьными и строковыми. В программе Hello World мы пользовались строковыми литералами. Это — последовательность символов, заключенная в двойные кавычки.

Символьный литерал служит для представления одиночного знака. Это символ, заключенный в одиночные кавычки (апострофы).

Числовые литералы могут быть вещественными (с плавающей точкой) и целыми. В случае целого литерала он может быть записан в десятичной, восьмеричной или шестнадцатеричной нотации (системе счисления). Вещественный литерал записывается либо в обычной десятичной, либо в экспоненциальной нотации. Наличие или отсутствие десятичной точки определяет, является ли константа вещественной или целой.

В таблице перечислены все упомянутые выше виды литеральных констант и даны соответствующие примеры.

***Таблица Литеральные константы***

|  |  |  |
| --- | --- | --- |
| **Литерал** | **Описание** | **Примеры** |
| Символьный | Одиночный символ, заключенный в апострофы | 'W', '&', 'Ф' |
| Строковый | Последовательность символов, заключенная в обычные (двойные) кавычки | "Это строка \n" |
| Целый | Десятичный — последовательность цифр, не начинающаяся с нуля | 123, 1999 |
|  | Восьмеричный — последовательность цифр от нуля до семерки, начинающаяся с нуля | 011, 0177 |
|  | Шестнадцатеричный — последовательность шестнадцатеричных цифр (0 - 9 и А - F), перед которой стоит 0X или Оx | ОХ9А, Oxffff |
| Вещественный | Десятичный — [цифры].[цифры] | 123., 3.14, .99 |
|  | Экспоненциальный — [цифры]Е|е[+|-] цифры | Зе-10, 1.17е6 |

Если константа заканчивается буквой L(l), то это длинная константа, для неё отводится участок памяти в 2 раза больше.

Можно дать литеральной константе некоторое имя, определив ее в качестве макроса препроцессора. После этого можно вместо литерала использовать имя. Это особенно удобно в том случае, когда одна и та же константа встречается в различных частях программы; используя имя вместо литералов, вы гарантированы от опечаток и, кроме того, гораздо проще вносить в код изменения, если значение константы нужно модифицировать. Макросы определяются директивой препроцессора **#define:**

**#define PI 3.14159265**

**#define TRUE 1**

При обработке исходного кода препроцессором выполняется просто текстовая подстановка: каждое вхождение имени макроса заменяется соответствующим ему литералом. Макросы называют также **символическими константами** (не путайте с символьными).

Типизированные константы

Разновидностью переменных являются типизированные константы. Это переменные, значение которых (заданное при инициализации) нельзя изменить. Создание типизированной константы ничем не отличается от инициализации переменной, за исключением того, что перед оператором объявления ставится ключевое слово **const**:

**const тип имя\_константы = значение [, ...];**

Например:

**const double Pi = 3.14159265;**

Ранее мы демонстрировали определение символической константы:

**#define PI 3.14159265**

Чем этот макрос отличается от показанной выше типизированной константы? Здесь следует иметь в виду два момента. Во-первых, типизированная константа по своему смыслу относится к конкретному типу данных, поэтому компилятор генерирует совершенно определенное представление для ее значения. Представление символической константы не определено.

Во-вторых, имя символической константы значимо только на этапе препроцессор ной обработки исходного кода, поэтому компилятор не включает ею в отладочную информацию объектного модуля. Вы не можете использовать это имя в выражениях при отладке. Напротив, типизированные константы являются по существу переменными, и их имена доступны отладчику. В силу этих причин предпочтительнее применять для представления постоянных величин типизированные константы, а не макросы **#define.**

**Структура программы**

По своей структуре программа на языке С представляет собой последовательный набор подпрограмм и функций одна из которых является основной программной единицей. Она начинается с заголовка **main()** или **main(<список параметров>).** Параметры могут задаваться, если программа через них производит связь с внешней средой. Параметры задаются идентификаторами через запятую. Если параметры есть, то они сразу должны быть описаны после заголовка. Затем идет { за которой следует описание переменных используемых в программе и операторы. Программная единица завершается}. На одной строке может размещаться любое количество операторов. Текст каждого оператор завершается ;. Комментарии записываются /\* текст \*/.

**Операторы языка С**

Оператор – это законченная инструкция, которая приказывает компилятору выполнить какое – либо действие. Операторы всегда заканчиваются ;

**X = 2 + 3 ;** оператор присваивания

Пустой оператор

Если поставить в отдельной строке ; - получится пустой оператор. Действий он не выполняет.

Составной оператор

Блок из двух и более операторов, заключенных в { } скобки называется составным опеатором

**{**

**printf("Hello,");**

**printf("World!!!");**

**}**

Выражения

Выражения – это любое сочетание элементов, которое в итоге образует числовое значение. Выражения могут быть различной сложности. Выражение записывается стандартным образом с использованием операндов, операций и круглых скобок. Значения вычисляются слева на право с учетом старшинства операций.

**X = 2 + a**

Здесь переменной Х присваивается выражение **2 + a ,** но и весь оператор присваивания является выражением

Вычисляемое значение

**Переменная = любое выражение**

Равно этому же значению

Допускаются выражения типа:

**y = x = a + 10;**

**x= 6 + (y = 4 + 5); (тогда** y=9 x=15**)**

**Операции**

Арифметические операции

Арифметические операции делятся на 2 одноместные и 5 двуместных.

1. Инкремент – увеличение операнда на единицу

**++X X++**

1. Декремент - уменьшение операнда на единицу

**- - X X - -**

Пример

**x = 10; y = x++; (x=11 y=10)**

**x = 10; y = ++ x; (x=11 y=11)**

1. Сложение + **(x + y)**
2. Вычитание – **(x - y)**
3. Умножение \* **(x \* y)**
4. Деление /**(x / y)**
5. Деление по модулю % **(x % y)**

Приоритет арифметических операций

1. ++, - -
2. \*, /, %
3. +, -

Операции отношений

== равно, != неравно, >, >=, <, <= Приоритет операций

1. >,>=,<,<=

2. ==, !=

Логические операции

**&&** and и

**||** or или

**!** not не

Для обозначения логических значений в языке С отсутствует специальный тип. Считается, что если оператор имеет целое значение 0, то он имеет значение ложь, а 1 - истина.

Поразрядные логические операции

Эти операции применяются к операндам целого типа и характеризуются тем, что применяются к отдельным разрядам двоичного значения операндов. Где двоичная единица — истина, а 0 — ложь.

**& -** поразрядное «и»

**~ -** унарная операция отрицания

**^ -** поразрядная операция «или»

**<<** - сдвиг значения влево на заданное число разрядов, при этом младшие разряды заполняются нулями.

**>>** - сдвиг значения вправо на заданное число разрядов, при этом младшие разряды пропадают, а освободившиеся старшие заполняются нулями.

5<<2 осуществляется сдвиг двоичного кода значения 5 на 2 разряда влево (в результате получается 20). Т.о. сдвиг влево на 1 разряд эквивалентен умножению на 2.

М>>К осуществляется сдвиг двоичного кода значения М на К разряда вправо. Т.о. сдвиг вправо на 1 разряд эквивалентен делению на 2.

Составные операторы присваивания

Если тип левой части этих операций не совпадает с типом правой части, то тип правой части преобразуется в тип левой.

|  |  |  |
| --- | --- | --- |
| **X+=Y** | эквивалентно | х= х + у |
| **X-=Y** | эквивалентно | х= х - у |
| **X/=Y** | эквивалентно | х= х / у |
| **X\*=Y** | эквивалентно | х= х \* у |
| **X%=Y** | эквивалентно | х= х % у |

Операция выбора по условию

Это единственная трехместная операция

**Ехр1 ? ехр2 : ехр3**

Если значение ехр1 равно true ( = 0*)* то всё выражение становится равным ехр2, если же ехр1 принимает значение false ( = 0), то выражение становится равным ехр3.

Х=у ? 1 : 100 (если у=0 то х=100 иначе х=1)

z=(х>у)?x:у (z присваивается наибольшее значение)

Запятая

Используется как знак препинания и как знак операции. При этом вычисляются оба выражения,

причем левое выражение первым, значением всего выражения считается результат вычисления

правого внутреннего выражения

Х=(a+ +, Ь++)

Значение в присваивается переменной x, после чего увеличивается a , затем увеличивается в. Чаще

всего запятая используется в операторе *for.*

Операция преобразования типа выражения

Эта операция вычисляет значение выражения, а затем результат преобразует в заданный тип.

**(имя типа) (выражение)**

Преобразование заключается в отбрасывании дробной части. Операция sizeof()

***Sizeof (аргумент)***

Где аргумент это имя переменной или константы или имя типа. В качестве результата выдается число байт памяти, занимаемое аргументом.

Приоритет операций

1. ( ) [] - >

2. **! ~ ++-- \* (ссылка по адресу) &(адрес)** *Sizeof* ***+ -* (одноместные)**

3. \* / % **(умножение)**

4. + -

*5*. << >>

6. < <= > >=

7. = = !=

8.  **&( побитовое и )**

9. ^

10. |

11. *&&*

12. | |

13. ? :

14.= += -= \*= /= %= *§=* ^= |= <<= >> =

15. *,* (**запятая**)