acm.mipt.ru

олимпиады по программированию на Физтехе

Раздел «Язык Си» . CfaqDeclaration :

Декларация и инициализация (краткий список)

- 10.1 Какой тип целочисленной переменной использовать
- 10.2 Каким должен быть новый 64-битный тип на новых 64-битных машинах?
- 10.3 У меня совсем не получается определение связанного списка.
- 10.4 Как объявить массив из N указателей на функции, возвращающие указатели на функции возвращающие указатели на char?
- 10.5 Я моделирую Марковский процесс с конечным числом состояний
- 10.6 Мой компилятор выдает сообщение о неверной повторной декларации, хотя я только раз определил функцию и только раз вызвал.
- 10.7 Как лучше всего декларировать и определить глобальные переменные?
- 10.8 Что означает ключевое слово extern при декларации функции?
- 10.10 Я видел, что функции вызываются с помощью указателей и просто как функции. В чем дело?

10.1 Какой тип целочисленной переменной использовать

Q: Какой тип целочисленной переменной использовать ?

A: Если могут потребоваться большие числа, (больше 32767 или меньше -32767), используйте тип long. Если нет, и важна экономия памяти (большие массивы или много структур), используйте short. Во всех остальных случаях используйте int. Если важно точно определить момент переполнения и/или знак числа не имеет значения, используйте соответствующий тип unsigned. (Но будьте внимательны при совместном использовании типов signed и unsigned в выражениях). Похожие соображения применимы при выборе между float и double.

Хотя тип char или unsigned char может использоваться как целочисленный тип наименьшего размера, от этого больше вреда, чем пользы из-за непредсказуемых перемен знака и возрастающего размера программы.

Эти правила, очевидно, не применимы к адресам переменных, поскольку адрес должен иметь совершенно определенный тип.

Если необходимо объявить переменную *определенного* размера, (единственной причиной тут может быть попытка удовлетворить внешним требованиям к организации памяти; см.,кроме того, вопрос 17.3), непременно изолируйте объявление соответствующим typedef.

Если вам нужны числа, большие чем гарантируют встроенные типы, или вам нужны вычисления с произвольной точностью (или "multiple precision"); см. вопрос 18.15d.

10.2 Каким должен быть новый 64-битный тип на новых 64-битных машинах?

Q: Каким должен быть новый 64-битный тип на новых 64-битных машинах?

A: Некоторые поставщики C компиляторов для 64-битных машин поддерживают тип long int длиной 64 бита. Другие же, опасаясь, что слишком многие уже написанные программы зависят от sizeof(int) == sizeof(long) == 32 бита , вводят новый 64-битный тип long long (или __longlong).

Программисты, желающие писать мобильные программы, должны, следовательно, изолировать 64-битные типы с помощью средства typedef. Разработчики компиляторов, чувствующие необходимость ввести новый целочисленный тип большего размера, должны объявить его как "имеющий по крайней мере 64 бит" (это действительно новый тип, которого нет в традиционном C), а не как "имеющий точно 64 бит".

10.3 У меня совсем не получается определение связанного списка.

Поиск
Поиск
Раздел «Язык Си»
Главная
Зачем учить С?
Определения
Инструменты:
Поиск
Изменения
Index
Статистика
Разделы
Информация
Алгоритмы
Язык Си
Язык Ruby
Язык Ассемблера
El Judge
Парадигмы
Образование
Сети
Objective C
Logon>>

Q: У меня совсем не получается определение связанного списка. Я пишу

```
typedef struct
{
     char *item;
     NODEPTR next;
} *NODEPTR;
```

но компилятор выдает сообщение об ошибке. Может ли структура в C содержать ссылку на себя?

A: Структуры в C, конечно же, могут содержать указатели на себя; обсуждение этого вопроса и пример в параграфе 6.5 K/R вполне проясняют этот вопрос. В приведенном тексте проблема состоит в том, что определение NODEPTR не закончено в том месте, где объявлется член структуры "next". Для исправления, снабдите сначала структуру тегом ("struct node"). Далее объявите "next" как "struct node *next;", и/или поместите декларацию typedef целиком до или целиком после объявления структуры. Одно из возможных решений будет таким:

```
  struct node
     {
     char *item;
     struct node *next;
     };

  typedef struct node *NODEPTR;
```

Есть по крайней мере три других одинаково правильных способа сделать то же самое. Сходная проблема, которая решается примерно так же, может возникнуть при попытке определить с помощью средства typedef пару ссылающихся друг на друга структур.

Смотри: K&R I Разд. 6.5 с. 101; K&R II Разд. 6.5 с. 139; H&S Разд. 5.6.1 с. 102; ANSI Разд. 3.5.2.3.

10.4 Как объявить массив из N указателей на функции, возвращающие указатели на функции возвращающие указатели на char?

Q: Как объявить массив из N указателей на функции, возвращающие указатели на функции возвращающие указатели на char?

A: Есть по крайней мере три варианта ответа:

1. `char *(*(*a[N])())();`

2. Писать декларации по шагам, используя =typedef:

```
typedef char *pc;   /* указатель на char */
typedef pc fpc();   /* функция,возвращающая указатель на char */
typedef fpc *pfpc;  /* указатель на.. см. выше */
typedef pfpc fpfpc();   /* функция, возвращающая... */
typedef fpfpc *pfpfpc;  /* указатель на... */
pfpfpc a[N];            /* массив... */
```

3. Использовать программу cdecl, которая переводит с английского на C и наоборот

```
  cdecl> declare a as array of pointer to function returning
      pointer to function returning pointer to char
  char *(*(*a[])())()
```

cdecl может также объяснить сложные декларации, помочь при явном приведении типов, и, для случая сложных деклараций, вроде только что разобранного, показать набор круглых скобок, в которые заключены аргументы. Версии cdecl можно найти в comp.sources.unix (см. вопрос 17.12) и в K&R II. Любая хорошая книга по C должна объяснять, как для понимания сложных деклараций, читать их "изнутри наружу", ("декларация напоминает использование").

Смотри: K&R II Разд. 5.12 с. 122; H&S Разд. 5.10.1 с. 116.

10.5 Я моделирую Марковский процесс с конечным числом состояний

Q: Я моделирую Марковский процесс с конечным числом состояний, и у меня есть набор функций для каждого состояния. Я хочу, чтобы смена состояний происходила путем возврата функцией указателя на функцию, соответветствующую следующему состоянию. Однако, я обнаружил ограничение в механизме деклараций языка C: нет возможности объявить функцию, возвращающую указатель на функцию, возвращающую указатель на функцию, возвращающую указатель на функцию...

A: Да, непосредственно это сделать нельзя. Пусть функция возвращает обобщенный указатель на функцию, к которому перед вызовом функции будет применен оператор приведения типа, или пусть она возвращает структуру, содержащую только указатель на функцию, возвращающую эту структуру.

10.6 Мой компилятор выдает сообщение о неверной повторной декларации, хотя я только раз определил функцию и только раз вызвал.

Q: Мой компилятор выдает сообщение о неверной повторной декларации, хотя я только раз определил функцию и только раз вызвал.

A: Подразумевается, что функции, вызываемые без декларации в области видимости (или до такой декларации), возвращают значение типа int. Это приведет к противоречию, если впоследствии функция декларирована иначе. Если функция возвращает нецелое значение, она должна быть объявлена до того как будет вызвана.

10.7 Как лучше всего декларировать и определить глобальные переменные?

Q: Как лучше всего декларировать и определить глобальные переменные?

A: Прежде всего заметим, что хотя может быть много *деклараций* (и во многих файлах) одной "глобальной" (строго говоря "внешней") переменной, (или функции), должно быть всего одно *определение*. (Определение - это такая декларация, при которой действительно выделяется память для переменной, и присваивается, если нужно, начальное значение). Лучше всего поместить определение в какой-то главный (для программы или ее части) .c файл, с внешней декларацией в головном файле .h, который при необходимости подключается с помощью #include. Файл, в котором находится определение переменной, также должен включать головной файл с внешней декларацией, чтобы компилятор мог проверить соответствие декларации и определения.

Это правило обеспечивает высокую мобильность программ и находится в согласии с требованиями стандарта ANSI C. Заметьте, что многие компиляторы и компоновщики в системе UNIX используют "общую модель", которая разрешает многократные определения без инициализации. Некоторые весьма странные компиляторы могут требовать явной инициализации, чтобы отличить определение от внешней декларации.

С помощью препроцессорного трюка можно устроить так, что декларация будет сделана лишь однажды, в головном файле, и она с помощью #define "превратится" в определение точно при одном включении головного файла.

Смотри: K&R I Разд. 4.5 с. 76-7; K&R II Разд. 4.4 с. 80-1; ANSI Разд. 3.1.2.2 (особенно Rationale), Разд. 3.7, 3.7.2, Разд. F.5.11; H&S Разд. 4.8 с. 79-80; CT&P Разд. 4.2 с. 54-56.

10.8 Что означает ключевое слово extern при декларации функции?

Q: Что означает ключевое слово extern при декларации функции? A: *Устарело*. слово extern при декларации функции может быть использовано из соображений хорошего стиля для указания на то, что определение функции, возможно, находится в другом файле. Формально между

```
extern int f();
```

и

```
int f();
```

нет никакой разницы.

Смотри: ANSI Разд. 3.1.2.2.

10.9 Я, наконец, понял, как объвлять указатели на функции, но как их инициализировать?

Q: Я, наконец, понял, как объвлять указатели на функции, но как их инициализировать?

A: Используйте нечто такое

```
extern int func();
int (*fp)() =   func;
```

Когда имя функции появляется в выражении, но функция не вызывается (то есть, за именем функции не следует "("), оно "сворачивается", как и в случае массивов, в указатель (т.е. неявным образом записанный адрес).

Явное объявление функции обычно необходимо, так как неявного объявления внешней функции в данном случае не происходит (опять-таки из-за того, что за именем функции не следует "(").

10.10 Я видел, что функции вызываются с помощью указателей и просто как функции. В чем дело?

Q: Я видел, что функции вызываются с помощью указателей и просто как функции. В чем дело?

A: По первоначальному замыслу создателя C указатель на функцию должен был "превратиться" в настоящую функцию с помощью оператора * и дополнительной пары круглых скобок для правильной интерпретации.

```
int r, func(), (*fp)() = func;
r = (*fp)();
```

На это можно возразить, что функции всегда вызываются с помощью указателей, но что "настоящие" функции неявно превращаются в указатели (в выражениях, как это происходит при инициализациях) и это не приводит к каким-то проблемам. Этот довод, широко распространенный компилятором pcc и принятый стандартом ANSI, означает, что выражение

```
r = fp();
```

работает одинаково правильно, независимо от того, что такое fp - функция или указатель на нее. (Имя всегда используется однозначно; просто невозможно сделать что-то другое с указателем на функцию, за которым следует список аргументов, кроме как вызвать функцию). Явное задание * безопасно и все еще разрешено (и рекомендуется, если важна совместимость со старыми компиляторами).

Смотри: ANSI Разд. 3.3.2.2 с. 41, Rationale с. 41.

10.11 Где может пригодиться ключевое слово auto?

Q: Где может пригодиться ключевое слово auto?

A: Нигде, оно вышло из употребления.