1. Понятие о структурных типах данных и о переменных этих типов
   1. Основные сведения

Структурный тип — это тип, каждая переменная, которого состоит из некоторого числа переменных других типов (как базовых так и раннее определенные структурных типов)

struct name\_type

{

тип\_компонента имя\_компонента;

тип\_компонента имя\_компонента;

.............................;

тип\_компонента имя\_компонента;

};

Пример:

struct point

{

int x;

int y;

};

Памяти выделилось 0

Инициализация переменной:

struct point

{

int x;

int y;

} vasia;

В си:

struct point vasia;

В с++:

point vasia;

* 1. Операции над переменными структурных типов

Операции над переменными структурных типов В С над переменными структурных типов разрешены присваивание, взятие адреса и осуществление доступа к её компонентам. Частным случаем этих операций является передача в функции или возврат из функции переменных структурных типов.

struct point makepoint(int x, int y)

{

struct point tmp;

tmp.x = x;

tmp.y = y;

return tmp;

}

Имена переменных и имена компонентов структурных типов могут совпадать при использовании в одной функции, так как они всегда различимы по контексту.

struct point makepoint(struct point p1, struct point p2)

{

p1.x += p2.x;

p1.y += p2.x;

return p1;

}

В ряде случаев более эффективно формой обеспечения доступа в функцию с

переменной структурных типов является передача указателей на такие

переменные.

При работе с компонентами структурных переменных с использованием

указателей обычно пользуются формой доступа:

указатель —> компонент

struct

{

int len;

char \*str;

} diskretka, \*proval=&diskretka;

++proval -> len;Оператор доступа к компонентам структурных типов вместе с механизмами

вызова функций и обращений к элементам массива имеют самый высокий

приоритет и выполняются раньше любых других операторов.

1.3 Массивы переменных структурного типа

Используя переменные структурных типов и, в особенности, массивы таких

переменных, средстами С/С++ можно решать задачи по обработке таблиц, т.е.

выполнять часть функций СУБД.

struct key

{

char \*word;

int count;

} massiv[100];

for (i=0;i<100;i++)

massiv[i].count++;

struct key \*elkin29;

for (elkin29=massiv;elkin29<massiv+100; elkin29++)

{

if (elkin29->count>0)

printf("%4d%s\n", elkin29->count, elkin29->word);

}

1.4 Объединения

Объединение — это переменная, которая в различные моменты времени

может хранить в себе объекты различных типов и размеров. Размер памяти,

выделяемой под объединение, таков, чтобы хватило места под любой его

компонент. Синтаксис работы с объединениями практически полностью

дублирует синтаксис работы со структурными типами и переменными этих

типов.

union poklonnik

{

int noginsk;

float \*obshaga;

}x;

В случае рассогласования типов(попытка обращения к компоненту

объединения в ситуации, когда последнее присвоенное значение

принадлежало другому компоненту) результат предсказать нельзя,

синтаксической ошибкой это не является.

Объединения могут входить в состав компонентов структурного типа и,

наоборот компонентами объединений могут быть переменные структурных

типов.

Struct

{

char \*name;

int reiting;

union

{

float vse;

int propalo;

} sovest;

} idb2205[100];

1.5 Поля битов

Поля битов — это некоторое множество битов, лежащих рядом внутри одного

машинного слова, длина которого зависит от реализаци. Синтаксис описания

полей битов и доступа к ним базируется на синтаксисе структурных типов.

struct vse\_ploho

{

unsigned int idb2201:1;

unsigned int idb2203:5;

unsigned int idb2205:1;

} st090301;

Числа следующие за двоеточиями задают ширину поля. Поля принято

описывать как беззнаковые целые(в теории они могут быть и просто целыми).

Компоненты принимают участие в операции как обычные целые, эти

компоненты не имеют отдельных адресов. Кроме того , компоненты не могут

быть массивами.

1.6 Средство typedef

В С/С++ существует возможность переименовать любой тип данных, как

базовый, так и определенный программистом.

typedef short unsigned int sovest;

typedef старое\_имя новое\_имя;

Использование typedef не создает новых типов, но повышает удобочитаемость программы за счет соответствия имен типов реалиям предметной области.

2. Особенности C++, отсутствующие в C.

2.1 Аргументы, используемые по умолчанию

В прототипе функции C++ можно указать значения по умолчанию для некоторых параметров. Это позволит пропускать часть аргументов при вызове функции, а на их место компилятор подставит указанные в прототипе значения. Параметры со значениями по умолчанию должны стоять после всех остальных параметров в списке. Если при вызове функции параметр опускается, то надо пропустить и все последующие параметры

void otchislenie (char\* gruppa, int x=2, int y=7)

otchislenie(“idb2205”);

otchislenie(“idb2201”, 0, 1);

otchislenie(1, 0); - неправильно, у gruppa нет значения по умолчанию

otchislenie(“idb2201”, 1);

2.2 Использование ссылок.

В C++ введен новый механизм под названием ссылка. Этот механизм позволяет возвращать из функций несколько переменных. Под ссылку не выделяется место в памяти, так как она является псевдонимом какой-то другой переменной.

Тип & имя;

int pank = 45;

int & roker = pank; - если ссылка описывается внутри модуля, то в строке описания сразу должно видно, чьим псевдонимом является ссылка. Чаще всего ссылки используются в качестве формальных параметров.

………..

Int oksana = 50;

f1(oksana);

f2(&oksana);

f3(&oksana);

……

1. void f1(int &idb2205)

{Idb2205++;}

2) Void f2 (int \*idb2205)

{ (\*idb2205)++;}

3) Void f3(int idb2205)

{ idb2205++; }

1 и 2 делают одно и то же, но вызов f1 не отличается от вызова f3, то есть ссылка позволяет передавать переменную по значению(синтаксически), но в виде ссылки.

Если тип возвращаемого значения – ссылка, то такую функцию можно вызывать слева от знака присваивания